2001-03

Implementation of Data Flow Query Language (DFQL)

Sahin, Ilker; Aksoy, Baybora

Monterey, California. Naval Postgraduate School

http://hdl.handle.net/10945/2772

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

IMPLEMENTATION OF DATA FLOW QUERY LANGUAGE (DFQL)

by

Baybora Aksoy
Ilker Sahin

March 2001

Thesis Advisor:	C.Thomas Wu
Second Reader:	Chris Eagle

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.			
1. AGENCY USE ONLY *(Leave blank)*	**2. REPORT DATE** March 2001	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE: Title (Mix case letters) **Implementation of Data Flow Query Language (DFQL)**			**5. FUNDING NUMBERS**
6. AUTHOR(S) Aksoy, Baybora Sahin, Ilker			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			**12b. DISTRIBUTION CODE**

13. ABSTRACT *(maximum 200 words)*

A relational database management system (RDBMS) is a software product that structures data in accordance with the relational data model and permits data manipulation based on relational algebra. There are two widely-used query languages for the relational database management systems (RDBMS's). These are Structured Query Language (SQL) and Query By Example (QBE). Although these languages are powerful, they both have drawbacks concerning ease-of-use, especially in expressing universal quantification and specifying complex nested queries. In order to eliminate these problems, Data Flow Query Language (DFQL) has been proposed. DFQL offers an easy-to-use graphical user interface to the relational model based on a data flow diagram, while maintaining all of the strengths of SQL and QBE.

The purpose of this thesis is to implement DFQL, allowing the users to login one or more relational database(s) through JDBC, view the structure of the connected databases graphically, and implement inquiries in SQL and DFQL to retrieve the data from the database(s).

14. SUBJECT TERMS Structure Query, SQL, QBE, Data Flow Query Language, DFQL, Java, JDBC, database structure			**15. NUMBER OF PAGES** 346
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

IMPLEMENTATION OF DATA FLOW QUERY LANGUAGE

Baybora Aksoy
Lieutenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1995

Ilker Sahin
Luitenant Junior Grade, Turkish Navy
B.S., Turkish Naval Academy, 1995

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2001

Author: [redacted]
Baybora Aksoy

Author: [redacted]
Ilker Sahin

Approved by: [redacted]
C.Thomas Wu, Thesis Advisor

[redacted]
Chris Eagle, Second Reader

[redacted]
Dan Boger, Chairman
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

A relational database management system (RDBMS) is a software product that structures data in accordance with the relational data model and permits data manipulation based on relational algebra. There are two widely-used query languages for the relational database management systems (RDBMS's). These are Structured Query Language (SQL) and Query By Example (QBE). Although these languages are powerful, they both have drawbacks concerning ease-of-use, especially in expressing universal quantification and specifying complex nested queries.

In order to eliminate these problems, Data Flow Query Language (DFQL) has been proposed. DFQL offers an easy-to-use graphical user interface to the relational model based on a data flow diagram, while maintaining all of the strengths of SQL and QBE.

The purpose of this thesis is to implement DFQL, allowing the users to login one or more relational database(s) through JDBC, view the structure of the connected databases graphically, and implement inquiries in SQL and DFQL to retrieve the data from the database(s).

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

We would like to thank the Turkish Navy for giving us the oppurtunity to study at the Naval Post Graduate School.

We would like to thank the Naval Post Graduate School for the outstanding study environment, rich academic resources, and distinguished faculty and staff. Especially, we would like to extend extra thanks to Proffessor Thomas C. Wu for his strong support, programming suggestions and guidance.

We are also very grateful to our friends, families, and most importantly, our wives Arzu Sahin and Berna Aksoy for their faith, support and patience.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

A. OVERVIEW

A relational database management system (RDBMS) is a software product that structures data in accordance with the relational data model and permits data manipulation based on relational algebra. There are two widely-used query languages for the relational database management systems (RDBMS's). These are Structured Query Language (SQL) and Query By Example (QBE). Although these languages are powerful, they both have drawbacks concerning ease-of-use, especially in expressing universal quantification and specifying complex nested queries.

In order to eliminate these problems, Data Flow Query Language (DFQL) has been proposed. DFQL offers an easy-to-use graphical user interface to the relational model based on a data flow diagram, while maintaining all of the strengths of SQL and QBE.

The purpose of this thesis is to implement DFQL, allowing the users to login one or more relational database(s) through Java Database Connectivity (JDBC), view the structure of the connected databases graphically, and implement inquiries in SQL and DFQL to retrieve the data from the database(s).

B. BACKGROUND

Relational Database technology was first described in 1970 by Ted Codd. His description of a database management system based on "relational" principles quickly turned from theory to real products which are known as Relational Database Management

Systems (RDBMS's). The most common query languages used to implement RDBMS products are Structured Query Language (SQL) and Query By Example (QBE).

The Structured Query Language (SQL) is a text-based language for accessing data in a relational database. Originally created by IBM in 1974, many vendors developed dialects of SQL. In the early 1980's, the American National Standards Institute (ANSI) began developing a relational database language standard. ANSI and the International Standards Organization (ISO) published SQL standards in 1986 and 1987, respectively [Ref.7:p.3].

On the other hand, Query By Example (QBE) is a visual-oriented language for accessing data in a relational database. QBE was developed by Zloof (1977) at IBM laboratories. Today, QBE is the best known and most widely-used visual RDBMS programming language [Ref.1:p.324].

Neither SQL nor QBE have succeeded in addressing the problems concerning ease-of-use issues, expecially in expressing universal quantification, specifying complex nested queries, flexibility and consistency in specifying queries with respect to data retrieval.

In order to alleviate these problems, a new visual-oriented language, Data Flow Query Language (DFQL) has been proposed. DFQL was first introduced in a research article entitled, "DFQL: Dataflow Query Language for Relational Databases" by Gard J. Clark, C. Thomas Wu, Naval Postgraduate School, Department of Computer Science.

Data Flow Query Language offers a graphical/visual interface to the relational model based on a data flow paradigm. DFQL retains all of the power of current query languages, and is equipped with an easy to use facility for extending the language with advanced operators, thus providing query facilities beyond the benchmark of first-order predicate logic [Ref.2:p.13].

C. CHAPTER SUMMARY

Chapter II presents a description of the Relational Model concept, SQL, QBE, and DFQL. The chapter also discusses the problems faced by SQL and QBE, and brings up DFQL solutions to these problems.

Chapter III presents the structure of our DFQL implementation. The chapter gives detailed information about the applications Database and Graphical User Interface.

Chapter IV specifies the system requirements and design details. This chapter also includes a user manual that shows the usage of all the features of the application and explains the purpose/function of them.

Chapter V offers conclusions about this work and recommendations for future work.

THIS PAGE INTENTIONALLY LEFT BLANK

II. RELATIONAL MODEL AND QUERY LANGUAGES

This chapter presents the relational model concept and the query languages used for the relational model. In order to understand why there is a need for a Data Flow Query Language, we need to study the relational model concept, analyze the other query languages, and identify the problems with them which can be mitigated through the use of DFQL.

The query languages for the relational database models can be classified as text-based query languages and visual-based query languages. Text-based query language part focuses on the Structured Query Language (SQL), while the visual-based query languages part focuses on Query By Example (QBE) and Data Flow Query Language (DFQL).

A. THE RELATIONAL MODEL

The Relational Model is very simple and elegant; a database is a collection of one or more relations, where each relation is a table with rows and columns. This simple tabular representation enables even novice users to understand the contents of a database, and permits the use of simple, high-level languages to query the data. The major advantages of the relational model over the older data models are its simple data representation and the ease with which even complex queries can be expressed [Ref.5:p.51].

The relational database model gives us the luxury of forgetting the actual physical data storage characteristics, thereby allowing us to concentrate on the logical view of the database. That is, we may focus on the human perception of data storage rather than on the often difficult-to-comprehend manner in which the computer sees those same data. Since the relational model achieves both **data independence** and **structural independence**, it becomes much easier to design the database and to manage its contents.

One of the reasons for the relational database model's rise to dominance in the database market is its very powerful and flexible query capability. [Ref.6:p.132].

B. TEXT-BASED QUERY LANGUAGES

Text-based query languages can be divided into three types: languages based on relational calculus, those based on relational algebra, and a combination of these two, which is SQL.

In relational calculus, a query describes the desired answer without specifying how the answer is to be computed. This non-procedural style of querying is called *declarative*. Queries in relational algebra are composed using a collection of operators, and each query describes a step-by-step procedure for computing the desired answer, (i.e., queries are specified in an operational manner.).

1. Structured Query Language (SQL)

The primary vehicle used for querying, reading, and updating relational databases is a language called Structured Query Language, or SQL (generally pronounced *sequel*). SQL is implemented as a mixture of both relational algebra and relational calculus by including the nesting capability and block structure feature.

Structured Query Language (SQL) is the most widely used commercial relational database language. SQL was originally developed at IBM in the SEQUEL-XRM and

System-R projects (1974-1977). Almost immediately, other vendors introduced DBMS products based on SQL, and is now a de facto standard. SQL continues to evolve in response to changing needs in the database arena. Although the new standard for SQL has recently been released (SQL-3), the most commonly used standard is still SQL-92.

SQL allows users to access data in relational database management systems, such as Oracle, Sybase, Informix, Microsoft SQL Server, Access, and others by allowing users to describe the data the user wishes to see. SQL also allows users to define the data (DDL) in a database, and manipulate that data (DML).

a. Strengths of SQL

The strengths of SQL provide benefits for all types of users, including application programmers, database administrators, managers, and end users. Technically speaking, SQL is a data sublanguage. Among features are the following:

- Processes sets of data as groups rather than as individual units.
- Provides automatic navigation to the data.
- Uses statements that are complex and powerful individually.
- Essentially, SQL lets you work with data at the level of logic. You need to be concerned with the implementation details only when you want to manipulate the data.

SQL provides statements for a variety of tasks, including:

- Querying data
- Inserting, updating, and deleting rows in a table
- Creating, replacing, altering, and dropping objects

- Controlling access to the database and its objects
- Guaranteeing database consistency and integrity

All major relational database management systems support SQL, so users can transfer all skills gained while using SQL from one database to another. In addition, all programs written in SQL are portable. They can often be moved from one database to another with very little modification.

b. Weaknesses of SQL

- **Difficulty in comprehending SQL:** SQL is primarily a declarative query language. It is very easy to express simple queries, but when it comes to more complex queries, the logical expression can become complicated. The lack of procedural nature of SQL is compensated by embedding it into a procedural, thire3rd generation host programming language [Ref.2:p.3].
- **Difficulty in expressing Universal Quantification:** The idea of universal quantification is expressed in English by the phrase "For All." There is no specific "For All" operator in SQL. This forces one to use "negative logic" (the Existential quantifier, NOT EXISTS to achieve the result of universal quantification) [Ref.2:p.3].
- **Lack of orthogonality:** SQL does not present a simple, clean, and consistent structure to the users. There are numerous examples of arbitrary restrictions, exceptions, and special rules. Lack of orthogonality increases

the number of special rules to be memorized by the user, decreases its readability, and in general decreases the usability [Ref.2:p.4].

- **Unnecessary complexity due to nesting construct:** SQL permits a nesting structure, which allows the specification of certain types of queries. These nesting structures are very hard to counstruct, and they increase the complexity of the expressions [Ref.2:p.4].

- **Lack of functional notation:** The use of functions in programming languages allows the abstraction of operational detail to whatever level is appropriate. Complex queries that provide an intermediate result for a higher level query could hide this result from the user through the use of functional notation. This concept is universally adopted in all modern programming languages, but not in SQL [Ref.2:p.4].

C. VISUAL-BASED QUERY LANGUAGES

Visual-based Query Languages are also used for querying relational data. These languages are different from text based query languages in that visual-based languages have a graphical user interface that allows users to construct queries. To define a query in the conventional database query language, a user must be familiar with the logical design of the database and the query language. With the recent advances in graphical user interfaces (GUI), visual query languages have been proposed to ease the user's burden in constructing a query.

The most important reason for having a visual query language for a scientific database application is to provide a simple yet integrated environment for data retrieval and visualization.

QBE and DFQL are examples of such visual query languages. They provide a graphical environment through which users construct queries and see the results. They are conceptually simple, expressive and helpful to user.

1. Query by Example (QBE)

Query By Example (QBE) was developed approximately at the same time as SQL during the 1970's at IBM's Laboratory Research Center [Ref.1:p.325]. QBE is another language for querying (and, like SQL, for creating and modifying) relational data. QBE is different from SQL and from most other database query languages in that QBE have a graphical user interface that allows users to write queries by creating sample tables on the screen. A user needs minimal information to get started, and the whole language contains relatively few concepts. QBE is especially suited for queries that are not too complex which can be expressed in a few tables.

a. Strengths of QBE

QBE is a very user-friendly query language with a graphical interface. This language provides a pictorial representation of database tables. Symbols placed in the proper table columns specify query selection conditions, grouping, data display, and database updates.

QBE is very intuitive, even for novice users. QBE allows the relatively inexperienced users to get started in specifying simple queries, even if they have no prior knowledge of programming languages.

b. Weaknesses of QBE

Although a very powerful tool for simple queries, QBE becomes less and less useful as the complexity of the queries increases, and its use presents problems with more complex queries [Ref.4:p.526].

QBE is relationally incomplete, since it does not provide support for universal or existential quantification. The queries which involve universal quantification can't be specified. [Ref.4:p.570]

2. Data Flow Query Language (DFQL)

DFQL is a visual relational algebra for the manipulation of relational databases. DFQL has sufficient expressive power and functionality to allow the user to express database queries.

Three principles guided our design of DFQL: conceptual simplicity, expressiveness, and provision of user assistance.

Because this language is targeted for users who are not programmers, DFQL should be conceptually simple. To achieve conceptual simplicity, the developed product focuses on the adoption of a functional orientation, maintaining compatibility with the relational data model and query operations, and use of understandable, explicitly stated operations in a query.

Since queries to a relational database are unconstrained in their form, our query language must be powerful enough to represent all queries a user may need. DFQL implements all the basic operators of relational algebra, and is therefore relational-complete. DFQL provides all of SQL's data retrieval capabilities as defined by the SQL

SELECT statement. DFQL also allows the user to *define* new operators in terms of external programs, and to integrate them into the language as new operators on data tables.

User assistance implies more than on-line documentation. A query should be visualized graphically, and assistance for query construction and troubleshooting and context-sensitive help should also be provided.

The operation logic of FDFQL is very simple. Data flows from one or more *source relations* to the final *result relation* through one or more intermediate *operators*. The result is constructed in the order specified by the directed acyclic graph. Every operator has one or more *input ports* and one *output port*. An arc can be attached to an operator only at these ports. An input port can only receive one arc. On the other hand there is no limit on the number of arcs that may emanate from an output port; each outgoing arc simply duplicates the output of the *operator*.

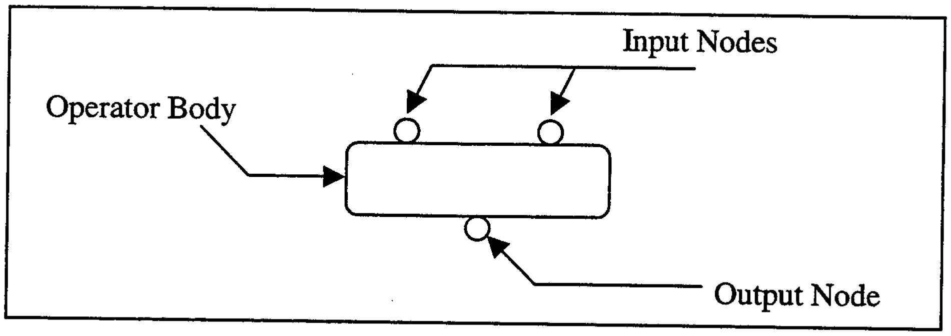


Figure 2.1: Operator Construction

A *source relation* in DFQL retrieves a user-specified relation from the underlying database.

The *result relation* may be thought of as a receptor for all the data that flow down from the sources of the graph. It provides a simple, plain text rendition of the data it receives. The result relation of each operator can be displayed to view the partial results without affecting the semantics of the graph being constructed.

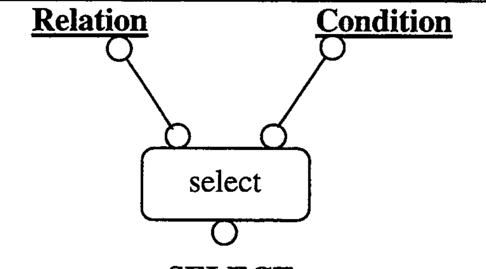
An *operator* receives one or more relations from its input arcs, processes them to produce a new relation, and then passes out the new relation. How the input relations are processed depends on the particular *operator*.

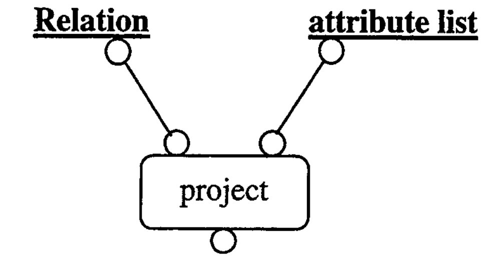
All operators of DFQL implement operational closure: output from each operator is always a relation. Two broad categories of DFQL operators are defined. A *primitive operator* has been defined directly in the native DFQL language. They are basic, non-basic (advanced) and display operators. A *user-defined operator* has been constructed by the user from primitives and possibly other previously created user-defined operators [Ref.2:p.6].

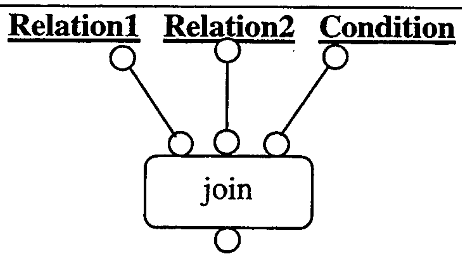
(1) Basic DFQL Operators: DFQL provides six basic operators derived from the requirements for relational completeness as well as form of grouping or aggregration. A query language that is complete has the expressive power of first-order predicate calculus. Five relational operations must be implemented: *selection, projection, union, difference, and Cartesian product (join).* Provision is also made for simple aggregation by including groupcnt (group count).

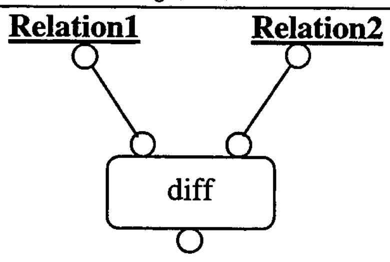
TABLE 2.1: Basic DFQL Operators

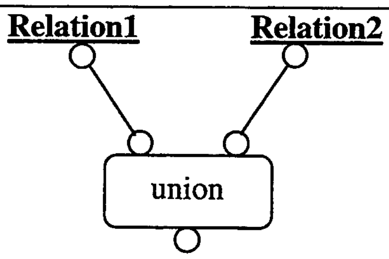
DFQL Operator	SQL Equivalent

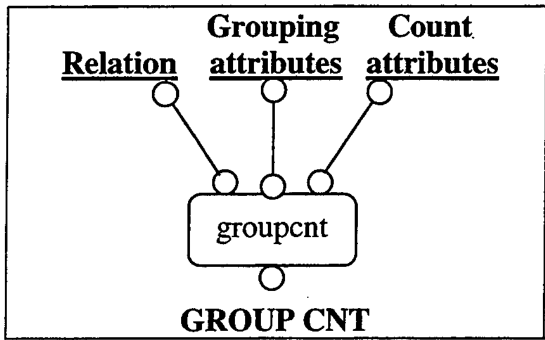

SELECT
	SELECT DISTINCT *
	FROM relation
	WHERE condition


PROJECT
	SELECT DISTINCT
	attribute list
	FROM relation


JOIN
	SELECT DISTINCT *
	FROM relation1 r1, relation2 r2
	WHERE condition


DIFFERENCE
	SELECT DISTINCT *
	FROM relation1
	MINUS
	SELECT DISTINCT *
	FROM relation2

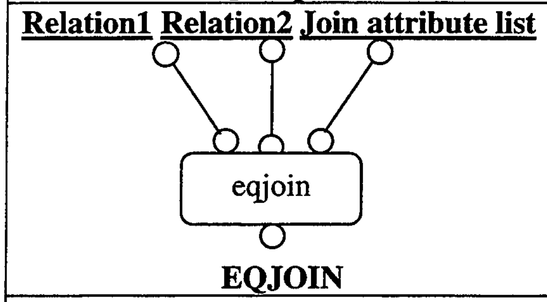

UNION
	SELECT DISTINCT *
	FROM relation1
	UNION
	SELECT DISTINCT *
	FROM relation2

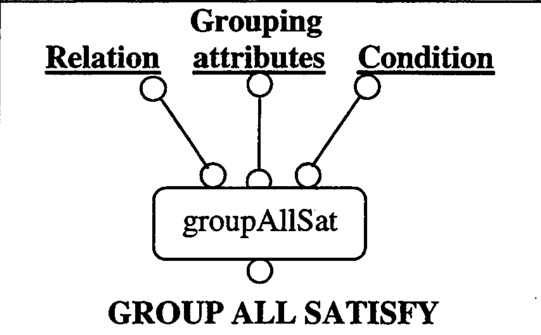
 Relation
Grouping attributes
Count attributes
groupcnt
GROUP CNT
	SELECT DISTINCT
	grouping attributes,
	COUNT(*) count attributes
	FROM relation
	GROUP BY grouping attributes

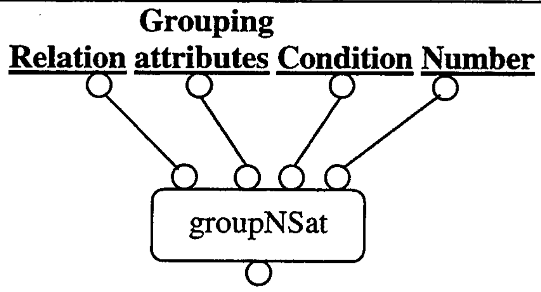
(2) Non-basic (Advanced) DFQL Operators: DFQL provides several other primitive operators to perform special low-level operations on relations. All of these primitive operators can be specified as user-defined operators.

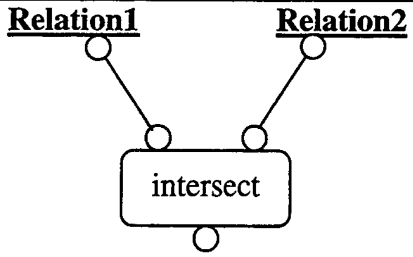
TABLE 2.2: Advanced DFQL Operators

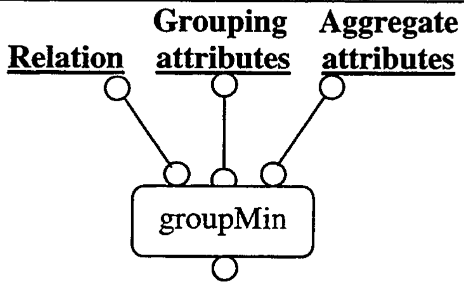
DFQL Operator	SQL Equivalent

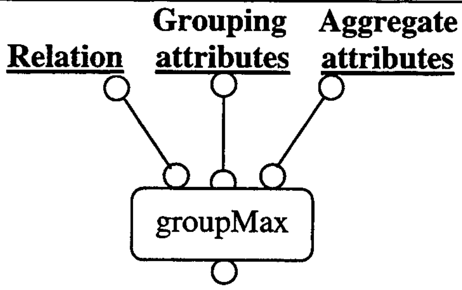
 Relation1
Relation2
Join attribute list
eqjoin
EQJOIN
	SELECT DISTINCT *
	FROM relation1 r1, relation2 r2
	WHERE r1.attribute = r2.attribute

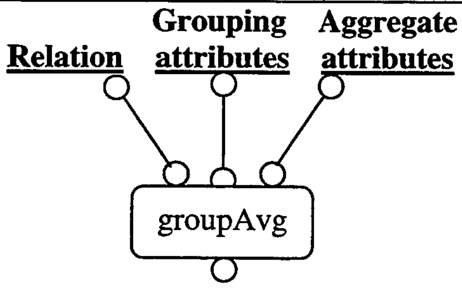
 Relation
Grouping attributes
Condition
groupAllSat
GROUP ALL SATISFY
	SELECT DISTINCT
	grouping attribute
	FROM relation
	WHERE condition
	GROUP BY grouping attribute

 Grouping
Relation
attributes
Condition
Number
groupNSat
GROUP N SATISFY
	SELECT DISTINCT
	grouping attribute
	FROM relation
	WHERE condition
	GROUP BY grouping attribute
	HAVING COUNT (*) number


INTERSECT
	SELECT DISTINCT *
	FROM relation1
	INTERSECT
	SELECT DISTINCT *
	FROM relation2

 Grouping
Aggregate
Relation
attributes
attributes
groupMin
GROUP MIN
	SELECT DISTINCT
	Grouping attributes,
	MIN (aggregate attributes)
	FROM relation
	GROUP BY
	Grouping attributes

 Grouping
Aggregate
Relation
attributes
attributes
groupMax
GROUP MAX
	SELECT DISTINCT
	Grouping attributes,
	MAX (aggregate attributes)
	FROM relation
	GROUP BY
	Grouping attributes

 Grouping
Aggregate
Relation
attributes
attributes
groupAvg
GROUP AVG
	SELECT DISTINCT
	Grouping attributes,
	AVG (aggregate attributes)
	FROM relation
	GROUP BY
	Grouping attributes

(3) Display Operators: The display operators are provided to allow the user to print the contents of a relation on the computer screen. The proposed implementation does not use display operators. Instead, one more control has been added to the operators. The default value for this control is *no display.* Checking this control displays the result of the respective operator.

(4) User-defined DFQL Operators: These types of operators provide the user with the flexibility to define his own style of the operators, and extend the capability of the language according to his desires. With user-defined operators, user can construct his own operators that look and behave like the primitive operators provided in DFQL.

THIS PAGE INTENTIONALLY LEFT BLANK

III. APPLICATION STRUCTURE

In this chapter, the structure of the overall application will be examined. There are two main parts in the implementation of DFQL, the *Database* and the *Graphical User Interface.*

A. DATABASE

This part of the application manages the connection with a JDBC database. There are five classes in the Database: Database, DatabaseInfo, Table, Attribute, and Relation classes.

The Database class is the interface between the graphical user interface and the JDBC database. This class holds all the data about the connected database. This data includes table data, connection information (database URL, driver name, user name, and password), database metadata, and tree structure of the database. The queries are executed through this class.

This class also has the capability to store all the queries executed in the current session and their results. The user can choose to store the queries and their results in the programs memory for future use. If a query is executed in the current session, this class does not send the query to the JDBC database but rather returns the result from the memory. Since the JDBC database can be manipulated by another source during the session, the application enables the user to oblige this class to send the query to the JDBC database.

The DatabaseInfo class holds the connection information. The driver name, database URL, user name and password data are held in this class. An instance of this class is a data member of the database object.

The Table class holds both the metadata and data of a relation. The information which can be stored in this class includes: the attribute information of a relation; the SQL query to get the relation; the relation name; and the user data. The relation can be a user defined relation of the JDBC database, or a resulting relation of a query.

The Attribute class holds the metadata of an attribute in a relation. This metadata contains the name, the data type, the size, the nullability, and the case sensitivity of an attribute.

The Relation class implements a JTable model in order to display the relation data on the screen.

B. GRAPHICAL USER INTERFACE

As previously mentioned, this language is targeted for non-programmer users; thus, the simplicity of the program is a very important factor. In order to ensure its simplicity, the graphical user interface (GUI) is designed to be very user friendly. GUI contains one main screen which allows the user to access every feature of the program as well as other screens for secondary operations, such as save/open file or print.

1. Main Screen

The main screen is the first screen displayed when the user runs the program. All the activities will occur on this screen. This screen contains a pull down menu, a toolbar, two scroll panels, two tabbed panels and two buttons. All of the main screen's features buttons, tabs, etc.) are in a disabled mode at the beginning. In order to enable them, the ser first needs to connect a database. The logon screen, which will be described later, is sed to make the database connections.

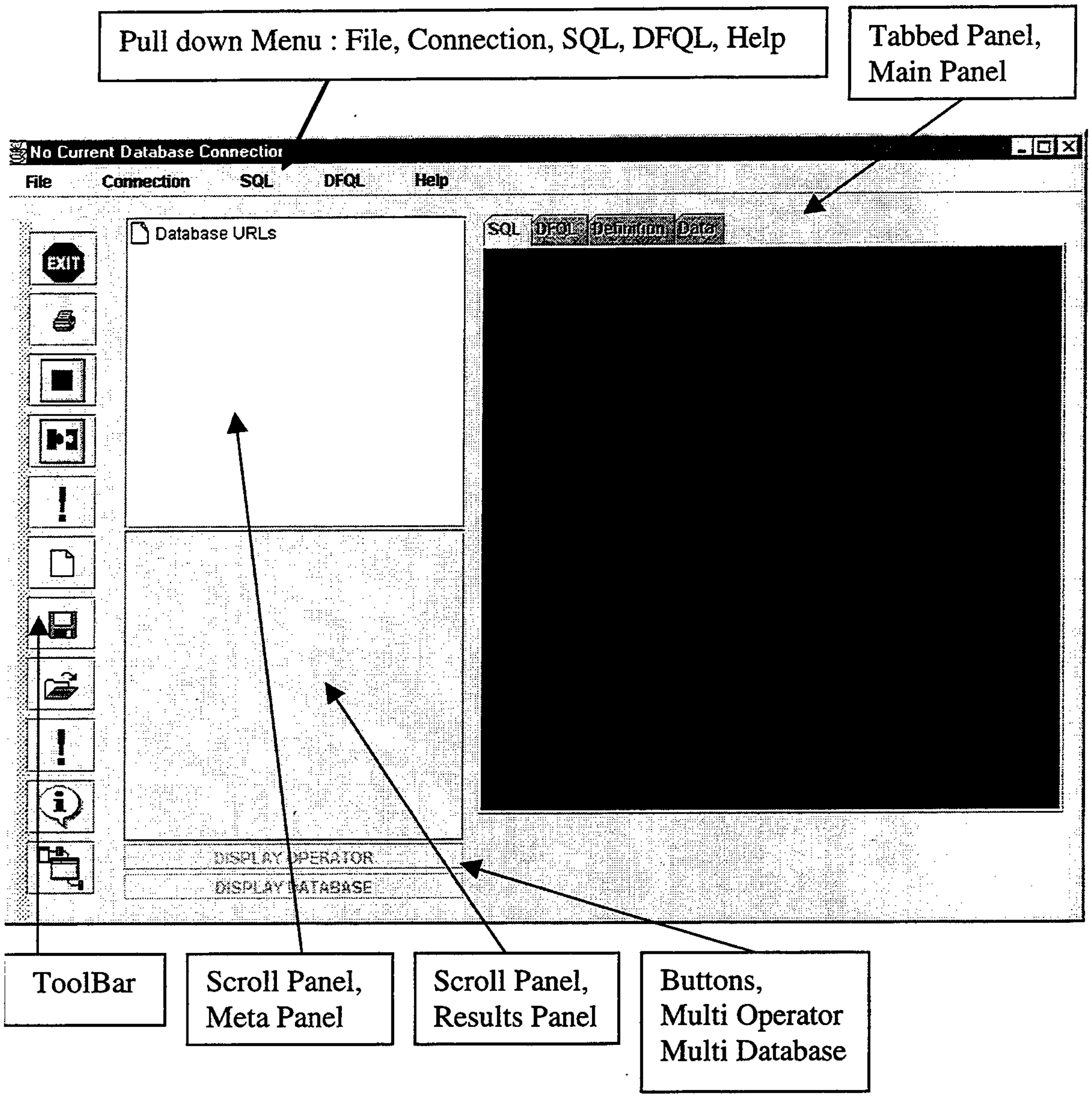


Figure 3.1: Main Screen

Pull Down Menu

- **File:** The file tab has two-sub menus, *Exit*, which terminates the program, and *Print,* which displays the print screen and allows user to print the results of queries.

- **Connection:** The connection tab has three sub-menus, C*onnect* (displays the logon screen), *Disconnect* (both are used to make database connections) and S*ettings*, which displays the settings screen and allows the user to arrange the connection settings.

- **SQL:** The SQL tab has only one sub-menu, *Execute,* which is used to execute the regular SQL query.

- **DFQL:** The DFQL tab has four sub-menus, *New, Save, Load* and *Execute.* New is used to clean all the previous DFQL work and prepare the program to create a new query. Save and Load are used to save a created DFQL query and reuse this query. Execute is used to run a DFQL query.

- **Help:** The help tab has only one sub-menu, *Manual,* which displays the user manual for this program.

ToolBar

The toolbar has eleven buttons. These buttons are: *Exit, Print, Connect, Disconnect, SQL Execute, New, Save, Load, DFQL Execute, Manual and Multiple database connection.* The purpose of these buttons is to provide fast access to the menu items, except the last one, database connection. This button is used to send the query to every database connected.

Panel Meta

This scroll panel is used to display a database's meta data tree. Panel holds a tree called a Meta tree. The root of this tree is a string, "Database URLs." Every database connected is added to this root. Each database has its relations, and each relation has its attributes as nodes. When the user disconnects a database, the related leaf of the tree is deleted.

Panel Results

This scroll panel is used to display the results of the executed query.

Buttons

There are two buttons on the main screen. These are *Display Operator* and *Display Database.* The Display operator button is used to display the result of the query, which is produced by the operator, whose display option is checked. Every time the user presses this button, the next operators query result is displayed. The Display Database button is used to send the same query to the next database connected.

Let us assume we have a DFQL query with two operators, and both of their display options are checked while connected to two databases at that moment. When the user presses the DFQL Exeute button, the first operator's query is sent to the first database connected. Then, the user presses the display operator button to see the result of the next operator. After that, the user presses the display database button to see the result of the first operator using the second database. And by pressing the display operator button, the second operator's query is sent to the second database.

Panel Main

This is a tabbed panel with four tab options. These are: *Data, Definition, DFQL* and *SQL.*

The data tab is used to display the content of the selected relation. When the user clicks on a relation with the left mouse button, this tab automatically becomes active and displays the data, which is held by this relation.

The definition tab is used to display detailed information about the database tree. This tab displays the properties of the selected item from the tree.

The SQL tab is basically a text editor which allows the user to enter a regular SQL query. This tab also displays the SQL query equivalent of the graphical DFQL query when one has been created.

The DFQL tab is a canvas which allows the user to create a DFQL query by using the built-in or user-defined operators. When this tab is selected, another tabbed panel is displayed automatically. This panel is called the *Operator's Panel,* and has three tabs. These are: *Basic, Advanced* and *User Defined.*

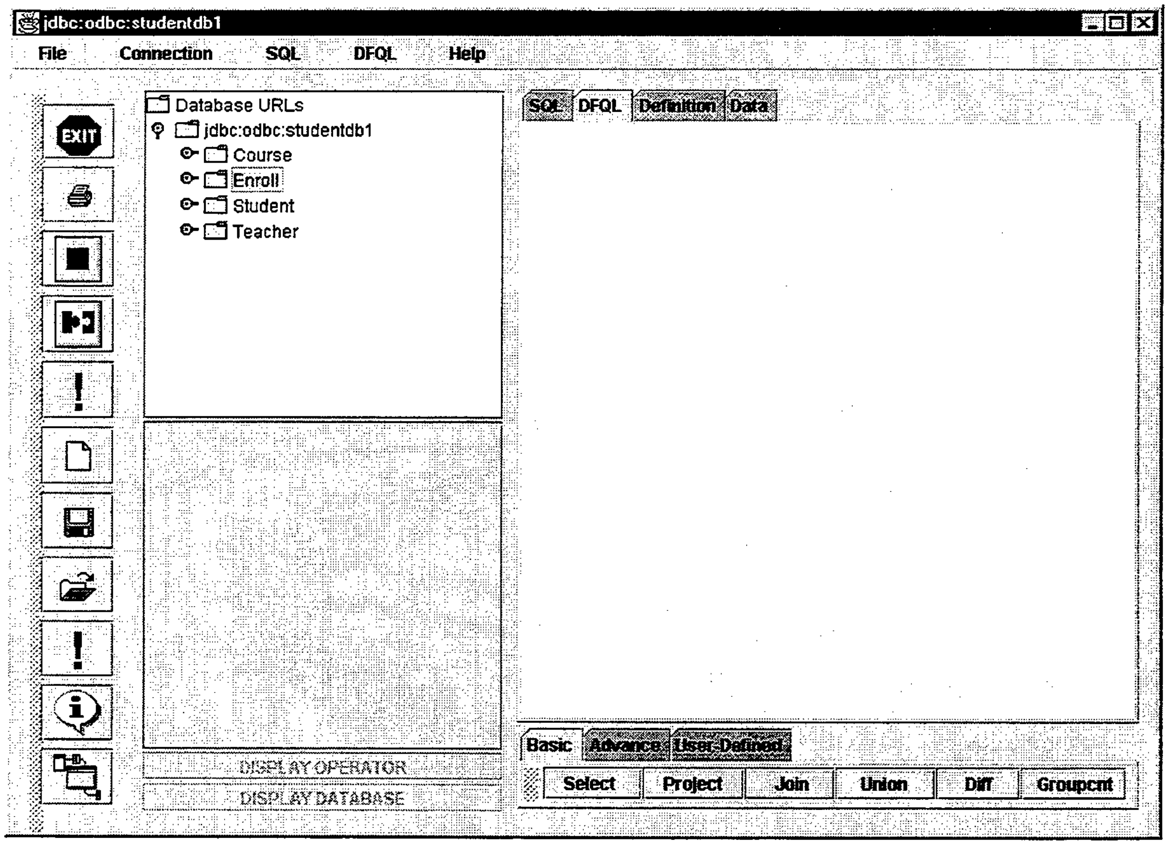
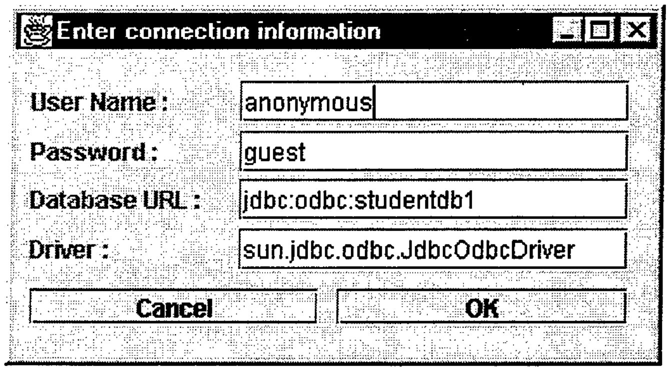


Figure 3.2: DFQL Panel and Operator Panel

2. Logon Screen

This screen allows the user to enter the connection information (User name, Password, Database URL and Database Driver), in order to log onto the target database.

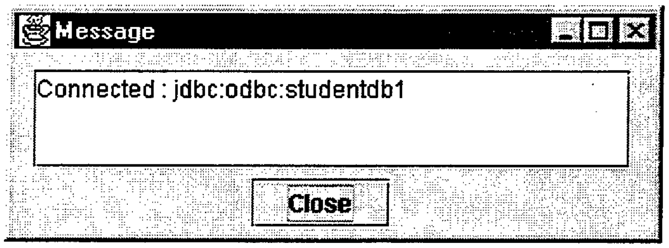


User Name : anonymous
Password : guest
Database URL : jdbc:odbc:studentdb1
Driver : sun.jdbc.odbc.JdbcOdbcDriver

Figure 3.3: Logon Screen

3. Message Screen

This screen is used to display system messages, such as connection information or error messages.

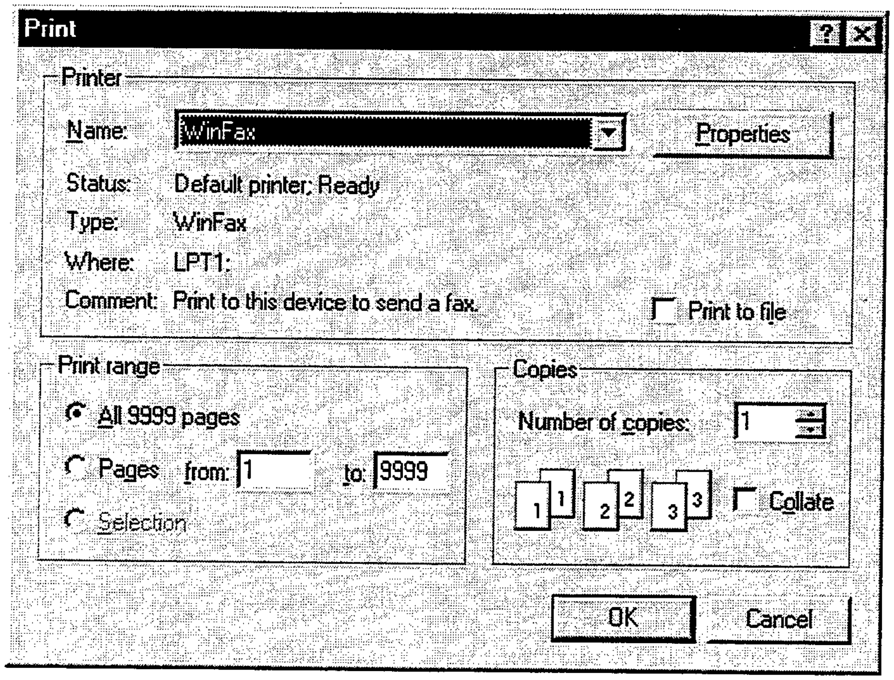


Connected : jdbc:odbc:studentdb1

Figure 3.4: Message Screen

4. Print Screen

This screen allows the user to print out the results of the executed query.


Print

Printer

Name: WinFax

Status: Default printer; Ready

Type: WinFax

Where: LPT1:

Comment: Print to this device to send a fax.

[ ] Print to file

Print range

[● All 9999 pages]

○ Pages from: 1 to: 9999

○ Selection

Copies

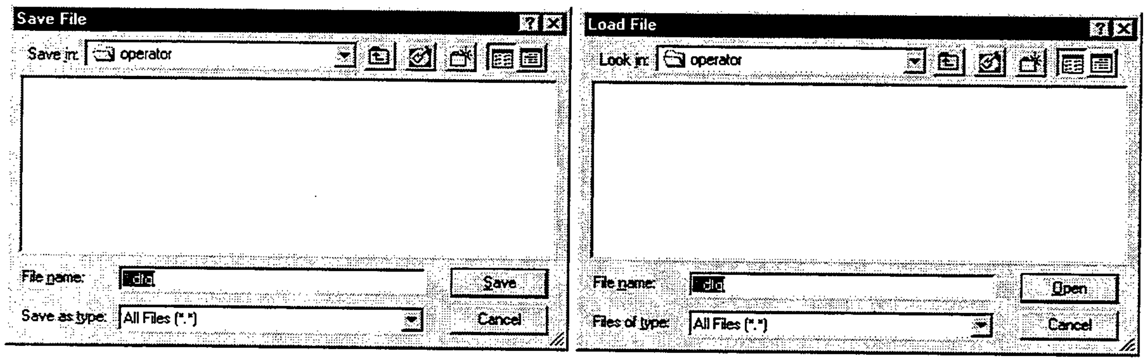
Number of copies: 1

[ ] Collate

Figure 3.5: Print Screen

5. Open/Save File Screen

This screen has two purposes. One is to save and load the previously created queries, and the second is to save and load user defined operators. Saved query files have an extension "dfql," and user defined operators have an extension "uso."


Save File

Save in: operator

File name: [illegible]

Save as type: All Files (*.*)

Load File

Look in: operator

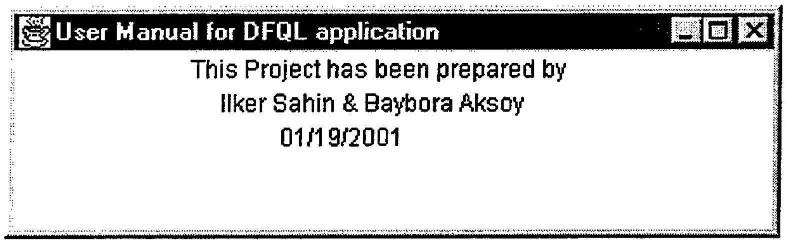
File name: [illegible]

Files of type: All Files (*.*)

Figure 3.6: Open and Save File Screens

6. User Manual Screen

This screen displays the user manual for this program. The user manual is a text file called "Manual.txt."


User Manual for DFQL application

This Project has been prepared by
Ilker Sahin & Baybora Aksoy
01/19/2001

Figure 3.7: User Manual Screen

THIS PAGE INTENTIONALLY LEFT BLANK

IV. IMPLEMENTATION OF APPLICATION

This chapter specifies the both hardware and software requirements for this application. It also gives the user more information about the design details. This chapter also includes a user manual that shows the usage of all the features of the application and explains the purpose/function of them.

A. HARDWARE REQUIREMENTS

- Intel Pentium/166MHz or compatible
- Microsoft Windows 95/98 or NT 4.0

  (SP3 or higher)(ODBC driver properly installed)
- 64Mb RAM (96Mb or higher recommended)
- 100Mb hard disk space
- SVGA or higher resolution (800x600 256 color)
- Mouse or other pointing device

B. SOFTWARE REQUIREMENTS

- Borland JBuilder 3.0
- Microsoft access 2000 or Oracle Relational Database Management System

  (Personal Oracle 8.x for windows98/NT)

C. SYSTEM SUMMARY

1. The user needs to provide database connection information to connect to any database. This information consists of the user's name, password, target database's URL and driver information.

2. After the connection has been successfully established, the connected database's meta data is displayed on the screen. This meta data consists of the names and the properties of the database, relations and relation attributes. Additionally, the content (data) of the relations is also displayed on the screen.

3. The user can concurrently connect to as many databases as desired. The last connected database is currently active, and the displayed Meta data belongs to this database. The user can change the active database by selecting the desired database.

4. The application has a text editor which allows the user to enter and execute a regular SQL statement. The result of this execution is also displayed on the screen.

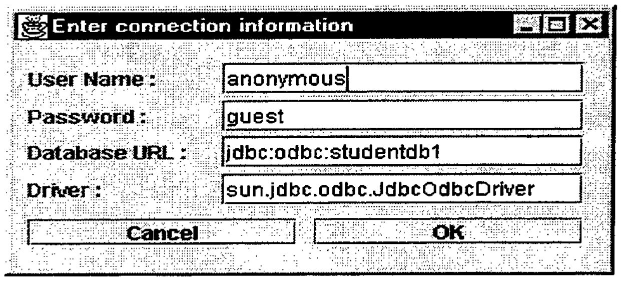
5. The user also can use DFQL operators to produce the data flow diagram graphically. The diagram automatically produces a query. This query is executed, and the result of this query is displayed on the screen.

6. Both SQL and DFQL queries can be executed for all the databases connected. The results are displayed in the order in which they were connected.

D. EXECUTION OF APPLICATION

1. Database Logon

In order to start using the program, the user must connect a database. The database connection can be done via a logon screen. The user can get the logon screen by pressing the connect button, or by selecting the connect menu item under the database menu. When the logon screen is displayed, the user needs to provide the following information for the desired database.


Enter connection information

User Name : anonymous

Password : guest

Database URL : jdbc:odbc:studentdb1

Driver : sun.jdbc.odbc.JdbcOdbcDriver

Figure 4.1: Database Logon

2. Multiple Database Connection

By using the same connection screen, the user can connect to as many databases as desired. Each connected database is added to the Meta data tree. The most recently connected database is assumed to be automatically active. The user can change the active database by selecting the database name from the database's Meta data tree.

3. Meta Data Tree

When the connection is completed, the database metadata is displayed on the screen. The left part of the screen displays the general database tree. The nodes under the database name are the relations of the database. The nodes under the relation nodes happen to be the attributes of the related relation. In order to obtain information about an item, the user simply needs to click on the item with the left mouse button.

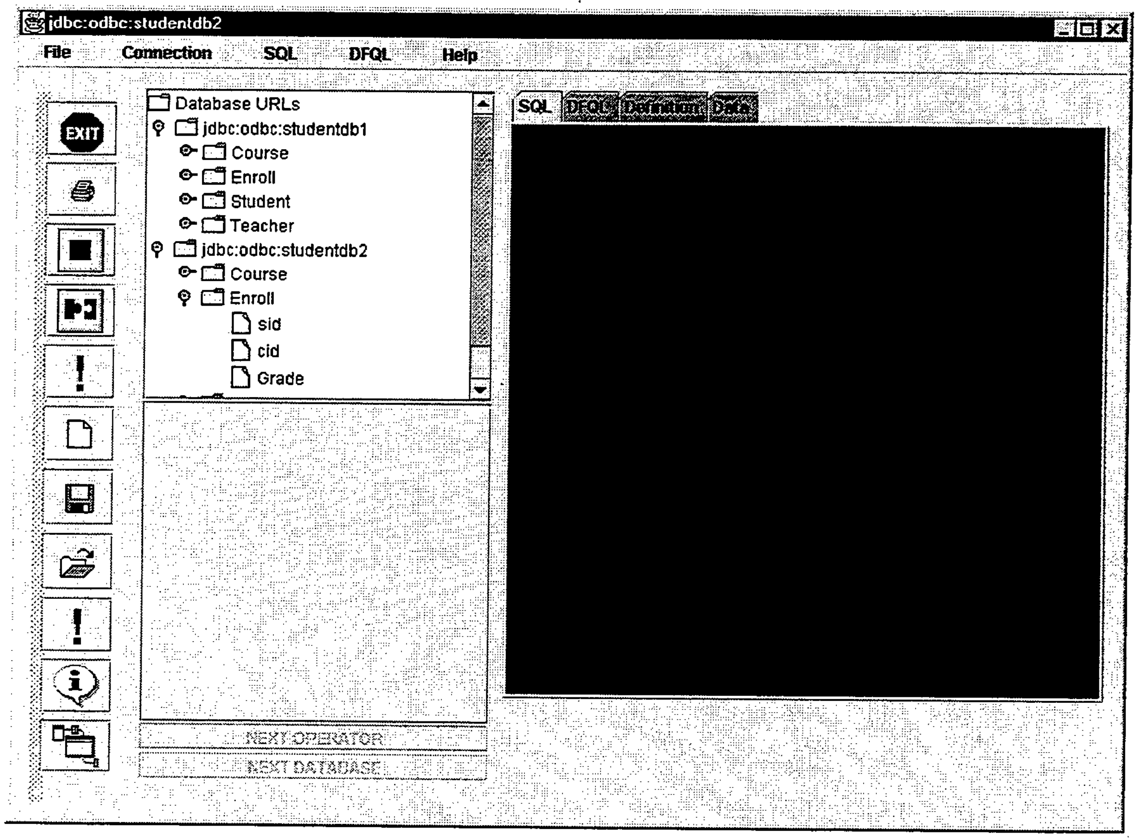
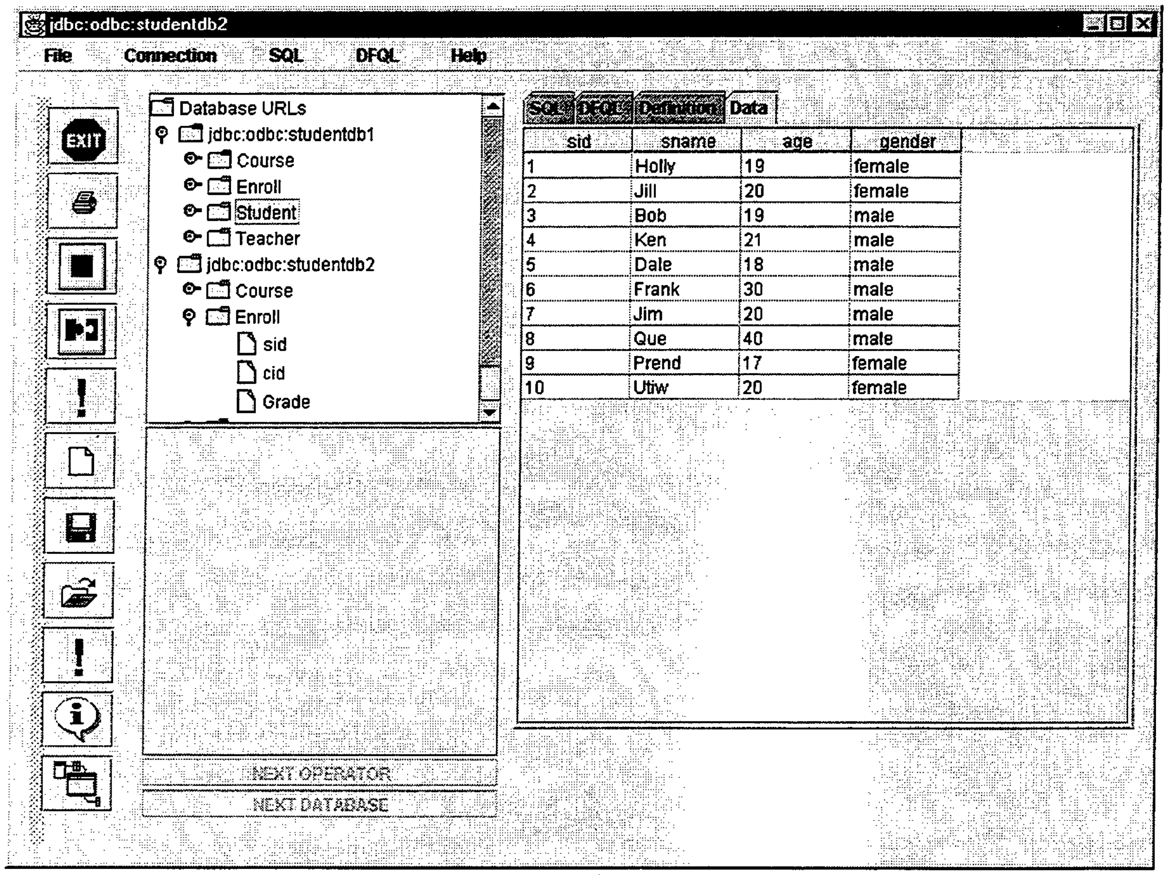


Figure 4.2: Meta Data Tree with Two Database Connection

4. Content of The Relations

The panel on the right side of the screen also displays the relation's content. In order to view the content of any relation, the user needs to select first the panel's data tab and then the relation desired.

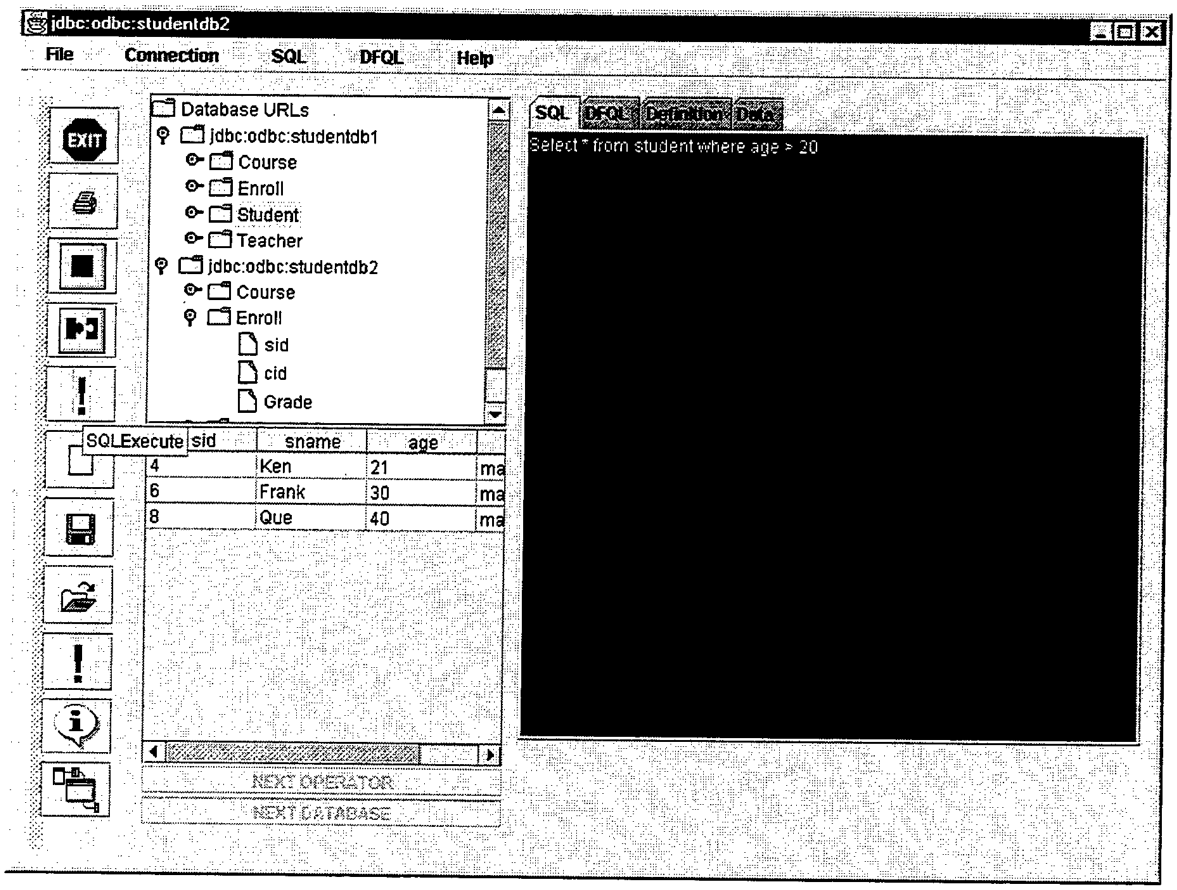


sid	sname	age	gender
1	Holly	19	female
2	Jill	20	female
3	Bob	19	male
4	Ken	21	male
5	Dale	18	male
6	Frank	30	male
7	Jim	20	male
8	Que	40	male
9	Prend	17	female
10	Utiw	20	female

Figure 4.3 : Content of the Relation, *Student*

5. Regular Query Execution

The right panel's SQL tab is used as a text editor for SQL queries. The user can enter a regular query on the text area and run this query by clicking the execute query button. The result of the query will show up on the result table. As an example, the query "**Select * from student where age>20**" is entered on the SQL text editor, is executed, and the result is displayed.



```
Select * from student where age > 20
```

sid	sname	age	
4	Ken	21	ma
6	Frank	30	ma
8	Que	40	ma

Figure 4.4: Regular Query Execution

6. Execution of Basic Operators

The user needs to select the main panel's DFQL tab to create a DFQL operator. There are six basic operators; these are, Select, Project, Join, Union, Difference and Group count. In order to create one of the six operators, the *basic panel* must be selected from the *operator panel*. After these selections are made, clicking on the empty canvas displays the basic operators list.

a. Execution of DFQL Operator - Select

The user selects the *select* operator from this list, or from the operator tab. After selection, the property window of this operator is displayed. The user gives this operator a name, enters a relation and a condition, and chooses a display mode. Then the user clicks the *OK* button to accept the input. Finally, from the *toolbar* or the *DFQL* menu, the user selects the execute button to run this operator.

For this example query, "**Select * from student where age>20**" is used.

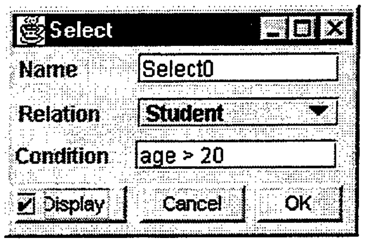
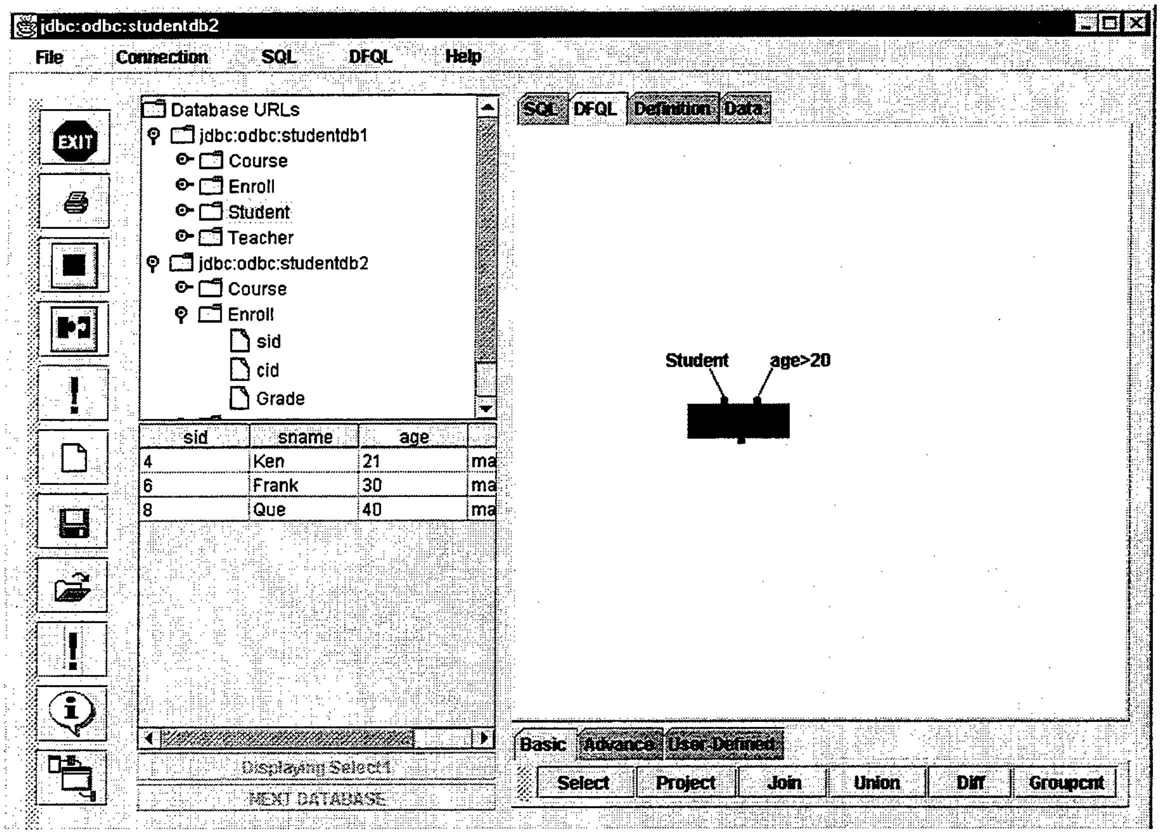


Figure 4.5: Operator Select Property Window



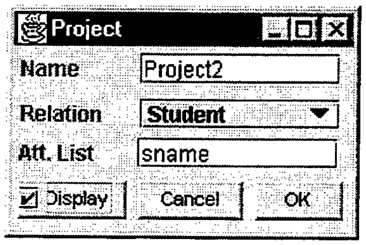
sid	sname	age	
4	Ken	21	ma
6	Frank	30	ma
8	Que	40	ma

Figure 4.6: Execution of Operator - Select

b. Execution of DFQL Operator – Project

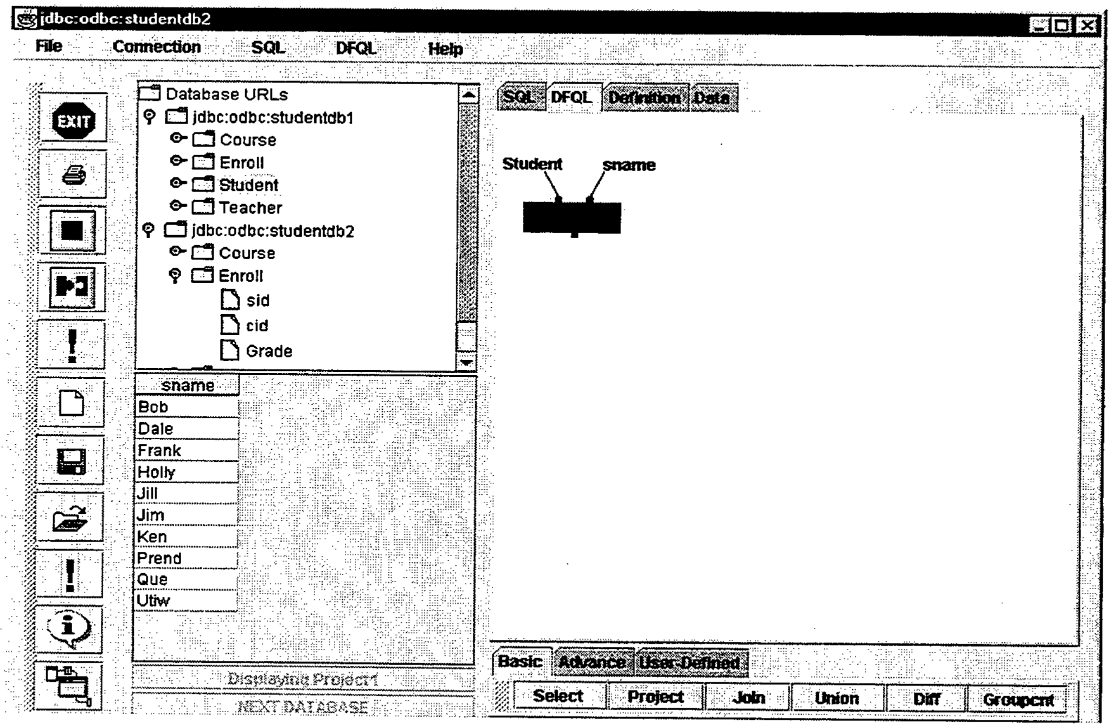
The user selects the *project* operator from this list, or from the operator tab. After selection, the property window of this operator is displayed. The user gives this operator a name, enters a relation and an attiribute, and chooses a display mode. Then the user clicks the *OK* button to accept the input. Finally, from the *toolbar* or the *DFQL* menu, the user selects the execute button to run this operator.

For this example query, "**Select sname from student**" is used.



Name: Project2
Relation: Student
Att. List: sname
Display: ✓

Figure 4.7: Operator Project Property Window



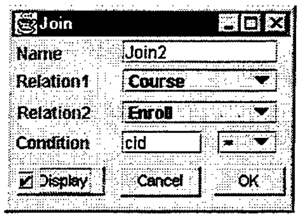
sname
Bob
Dale
Frank
Holly
Jill
Jim
Ken
Prend
Que
Utiw

Figure 4.8: Execution of DFQL Operator – Project

c. *Execution of DFQL Operator – Join*

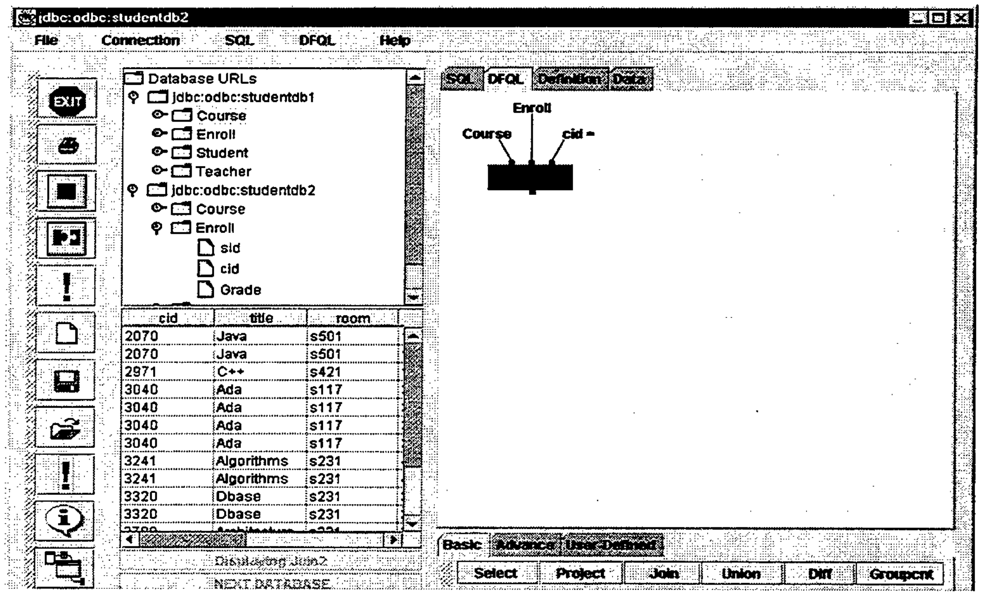
The user selects the *join* operator from this list, or from the operator tab. After selection, the property window of this operator is displayed. The user gives this operator a name, enters two relations, a condition, and a relation operator, and chooses a display mode. Then the user clicks the *OK* button to accept the input. Finally, from the *toolbar* or the *DFQL* menu, the user selects the execute button to run this operator.

For this example query, "**Select distinct * from Course c, Enroll e where c.cid = e.cid**" is used.


Join

Name	Join2
Relation1	Course
Relation2	Enroll
Condition	cid =

Figure 4.9: Operator Join Poperty Window



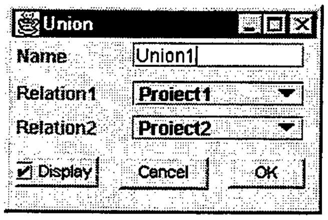
cid	title	room
2070	Java	s501
2070	Java	s501
2971	C++	s421
3040	Ada	s117
3040	Ada	s117
3040	Ada	s117
3040	Ada	s117
3241	Algorithms	s231
3241	Algorithms	s231
3320	Dbase	s231
3320	Dbase	s231

Figure 4.10: Execution of DFQL Operator – Join

d. *Execution of DFQL Operator – Union*

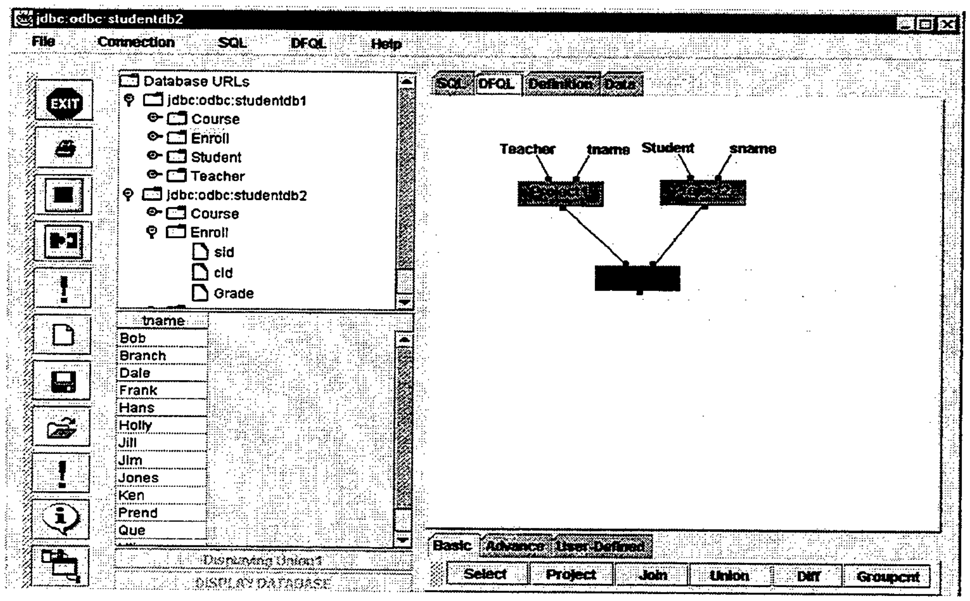
The user selects the *union* operator from this list, or from the operator tab. After selection, the property window of this operator is displayed. The user gives this operator a name, enters two relations, and chooses a display mode. Then the user clicks the *OK* button to accept the input. Finally, from the *toolbar* or the *DFQL* menu, the user selects the execute button to run this operator.

For this example query, "**Select distinct * from (select distinct tname from Teacher) union select distinct * from (select distinct sname from Student)**" is used.



Name: Union1
Relation1: Project1
Relation2: Project2
[✓ Display]

Figure 4.11: Operator Union Property Window



tname
Bob
Branch
Dale
Frank
Hans
Holly
Jill
Jim
Jones
Ken
Prend
Que

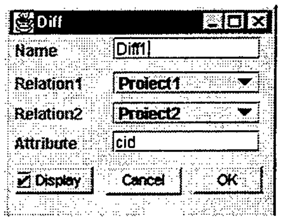
Displaying Union1

Figure 4.12: Execution of DFQL Operator – Union

e. Execution of DFQL Operator – Difference

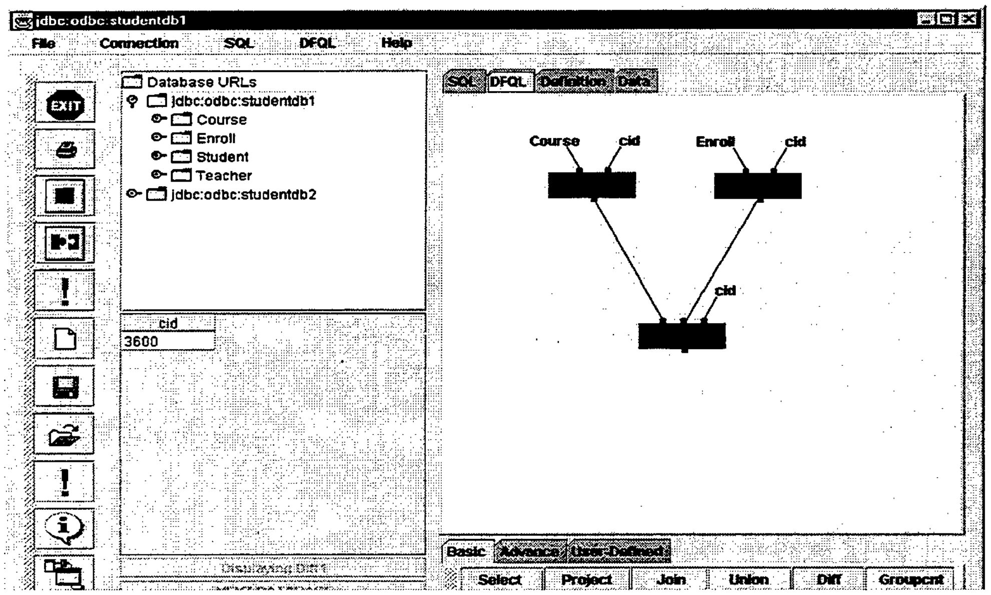
The user selects the *diff* operator from this list, or from the operator tab. After selection, the property window of this operator is displayed. The user gives this operator a name, enters two relations and an attribute for comparison, and chooses a display mode. Then the user clicks the *OK* button to accept the input. Finally, from the *toolbar* or the *DFQL* menu, the user selects the execute button to run this operator.

For this example query, "**Select distinct cid from (select distinct cid from Course) S where S.cid not in(select cid from (select distinct cid from Enroll))**" is used.



Name	Diff1
Relation1	Project1
Relation2	Project2
Attribute	cid

Figure 4.13: Operator Difference Property Window



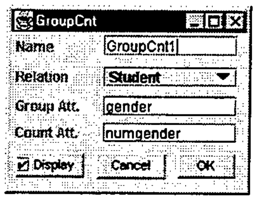
cid
3600

Figure 4.14: Execution of DFQL Operator - Diff

f. Execution of DFQL Operator – GroupCnt

The user selects the *groupCnt* operator from this list, or from the operator tab. After selection, the property window of this operator is displayed. The user gives this operator a name, enters a relation, a grouping attribute and a count attribute, and chooses a display mode. Then the user clicks the *OK* button to accept the input. Finally, from the *toolbar* or the *DFQL* menu, the user selects the execute button to run this operator.

For this example query, "**Select distinct gender, count (*) As numgender from Student group by gender**" is used.



GroupCnt

Name: GroupCnt1

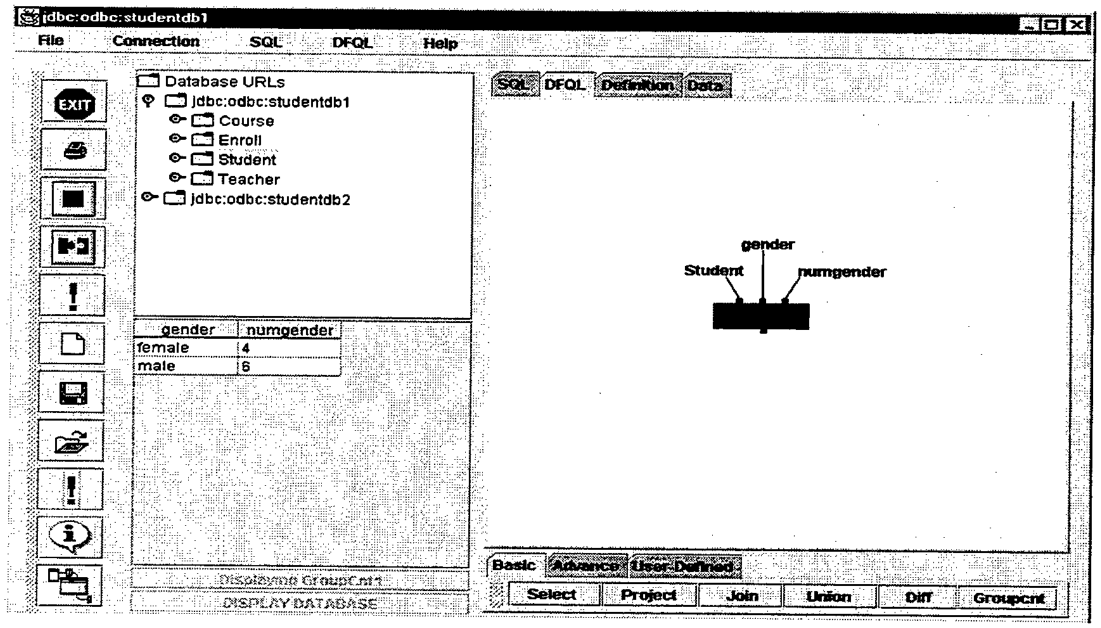
Relation: Student

Group Att.: gender

Count Att.: numgender

☑ Display

Figure 4.15: Operator GroupCnt Property Window



jdbc:odbc:studentdb1

gender	numgender
female	4
male	6

Displaying GroupCnt1

DISPLAY DATABASE

Figure 4.16: Execution of DFQL Operator – GroupCnt

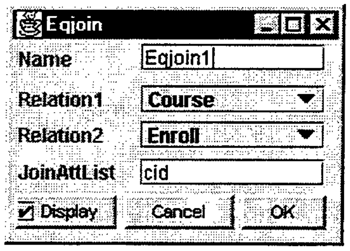
7. Execution of Advanced Operators

The user needs to select the main panel's DFQL tab to create a DFQL operator. There are seven advanced operators; these are Eqjoin, GroupAllSatisfy, GroupNSatisfy, GroupMax, GroupMin, GroupAvg and Intersect. In order to create one of the seven operators, the *advanced panel* must be selected from the *operator panel.* After these selections are made, clicking on the empty canvas displays the basic operators list.

a. Execution of DFQL Operator – Eqjoin

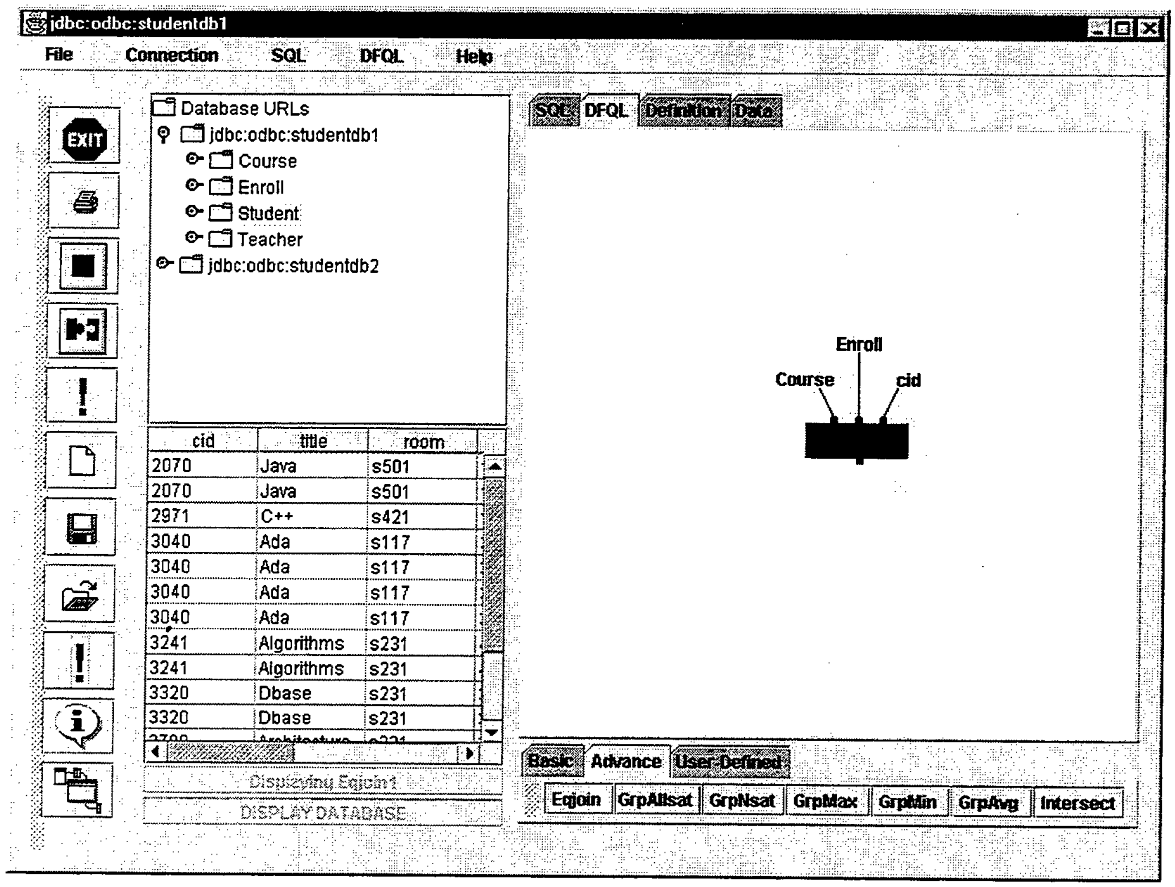
The user selects the *eqjoin* operator from this list, or from the operator tab. After selection, the property window of this operator is displayed. The user gives this operator a name, enters two relations and an attribute, and chooses a display mode. Then the user clicks the *OK* button to accept the input. Finally, from the *toolbar* or the *DFQL* menu, the user selects the execute button to run this operator.

For this example query, "**Select distinct * from Course s1,Enroll s2 where s1.cid = s2.cid**" is used.


Eqjoin

Name	Eqjoin1
Relation1	Course
Relation2	Enroll
JoinAttList	cid

Figure 4.17: Operator Eqjoin Property Window



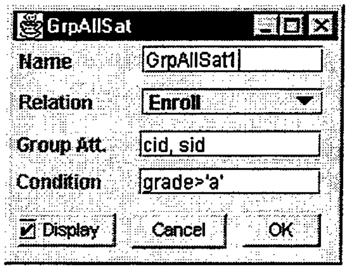
cid	title	room
2070	Java	s501
2070	Java	s501
2971	C++	s421
3040	Ada	s117
3040	Ada	s117
3040	Ada	s117
3040	Ada	s117
3241	Algorithms	s231
3241	Algorithms	s231
3320	Dbase	s231
3320	Dbase	s231

Figure 4.18: Execution of DFQL Operator – Eqjoin

b. *Execution of DFQL Operator – GroupAllSatisfy*

The user selects the *groupAllSat* operator from this list, or from the operator tab. After selection, the property window of this operator is displayed. The user gives this operator a name, enters a relation, a grouping attribute and a condition, and chooses a display mode. Then the user clicks the *OK* button to accept the input. Finally, from the *toolbar* or the *DFQL* menu, the user selects the execute button to run this operator.

For this example query, "**Select distinct cid, sid from Enroll where grade>'a' group by cid, sid**" is used.


GrpAllSat

Name: GrpAllSat1

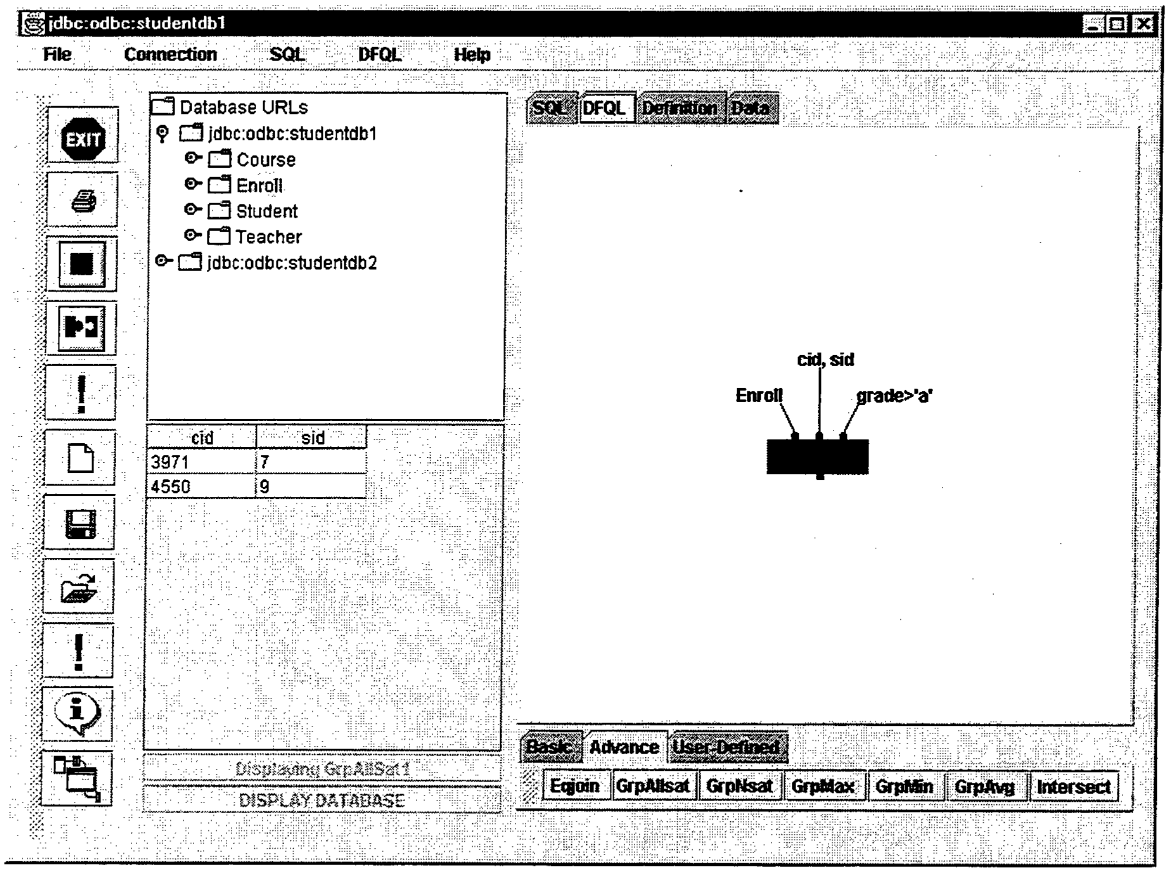
Relation: Enroll

Group Att.: cid, sid

Condition: grade>'a'

☑ Display

Figure 4.19: Operator GroupAllSatisfy Property Window



cid	sid
3971	7
4550	9

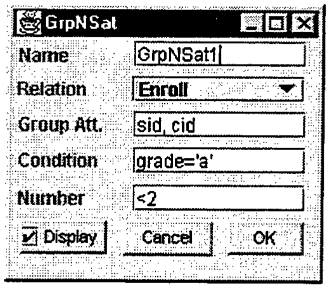
Displaying GrpAllSat1

Figure 4.20: Execution of DFQL Operator - GroupAllSat

c. *Execution of DFQL Operator – GroupNSatisfy*

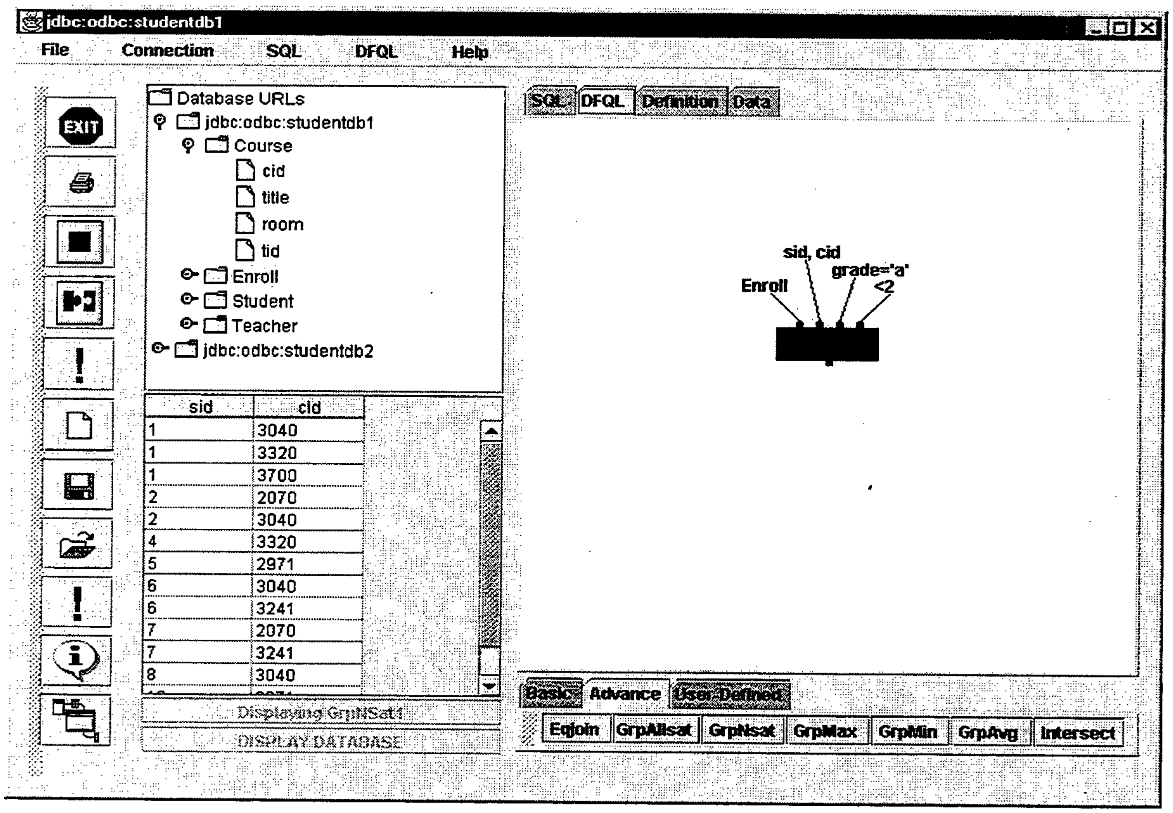
The user selects the *groupNSat* operator from this list, or from the operator tab. After selection, the property window of this operator is displayed. The user gives this operator a name, enters a relation, a grouping attribute and a condition, and chooses a display mode. Then the user clicks the *OK* button to accept the input. Finally, from the *toolbar* or the *DFQL* menu, the user selects the execute button to run this operator.

For this example query, "**Select distinct sid, cid from Enroll where grade='a' group by sid, cid having count (*) <2**" is used.



Name	GrpNSat1
Relation	Enroll
Group Att.	sid, cid
Condition	grade='a'
Number	<2

Figure 4.21: Operator GroupNSatisfy Property Window



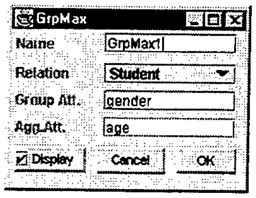
sid	cid
1	3040
1	3320
1	3700
2	2070
2	3040
4	3320
5	2971
6	3040
6	3241
7	2070
7	3241
8	3040

Figure 4.22 : Execution of DFQL Operator - GroupNSat

d. *Execution of DFQL Operator – GroupMax*

The user selects the *groupMax* operator from this list, or from the operator tab. After selection, the property window of this operator is displayed. The user gives this operator a name, enters a relation, a grouping attrribute and an aggragate attribute, and chooses a display mode. Then the user clicks the *OK* button to accept the input. Finally, from the *toolbar* or the *DFQL* menu, the user selects the execute button to run this operator.

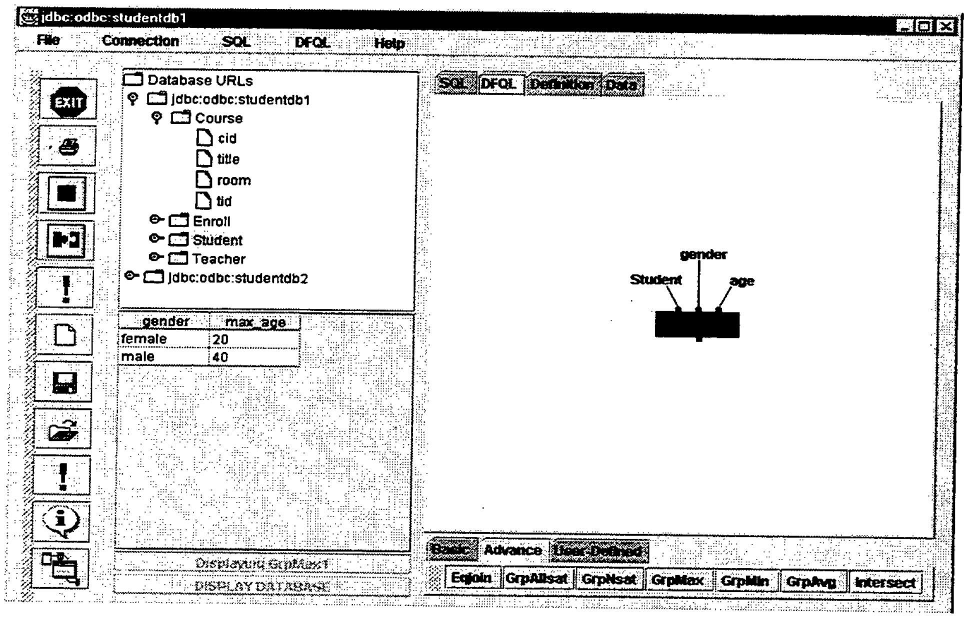
For this example query, "**Select distinct gender, max (age) as max_age from Student group by gender**" is used.



GrpMax

Name	GrpMax1
Relation	Student
Group Att.	gender
Agg.Att.	age

Figure 4.23: Operator GroupMax Property Window



jdbc:odbc:studentdb1

gender	max_age
female	20
male	40

Displaying GrpMax1

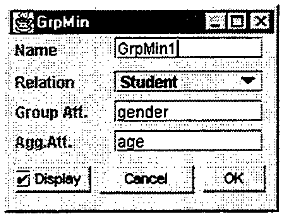
DISPLAY DATABASE

Figure 4.24: Execution of DFQL Operator - GroupMax

e. Execution of DFQL Operator – GroupMin

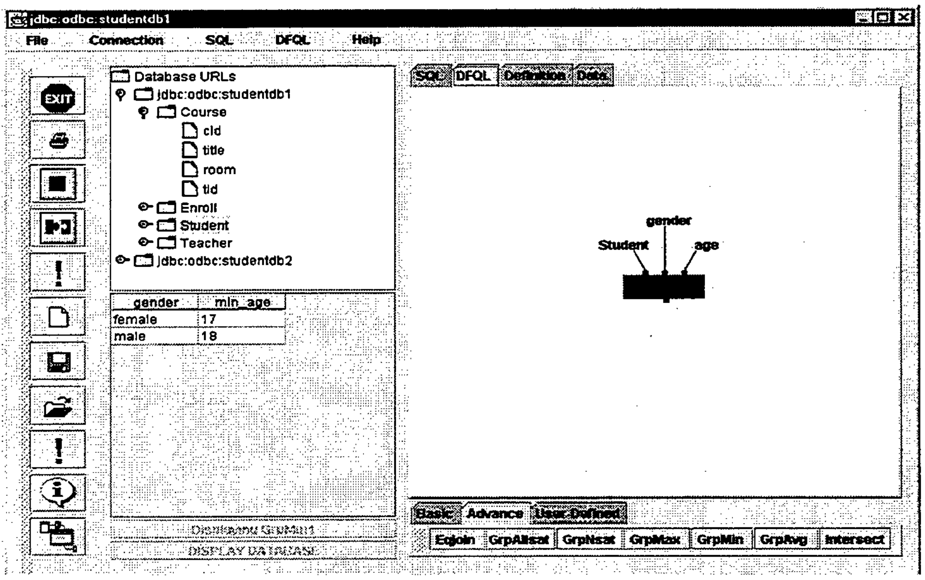
The user selects the *groupMin* operator from this list, or from the operator tab. After selection, the property window of this operator is displayed. The user gives this operator a name, enters a relation, a grouping attrribute and an aggragate attribute, and chooses a display mode. Then the user clicks the *OK* button to accept the input. Finally, from the *toolbar* or the *DFQL* menu, the user selects the execute button to run this operator.

For this example query, "**Select distinct gender, min (age) as min_age from Student group by gender**" is used.



Name	GrpMin1
Relation	Student
Group Att.	gender
Agg.Att.	age

Figure 4.25: Operator GroupMin Property Window



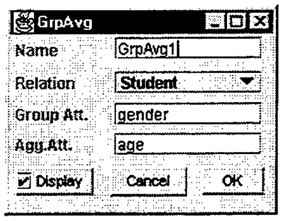
gender	min_age
female	17
male	18

Figure 4.26: Execution of DFQL Operator - GroupMin

f. Execution of DFQL Operator – GroupAvg

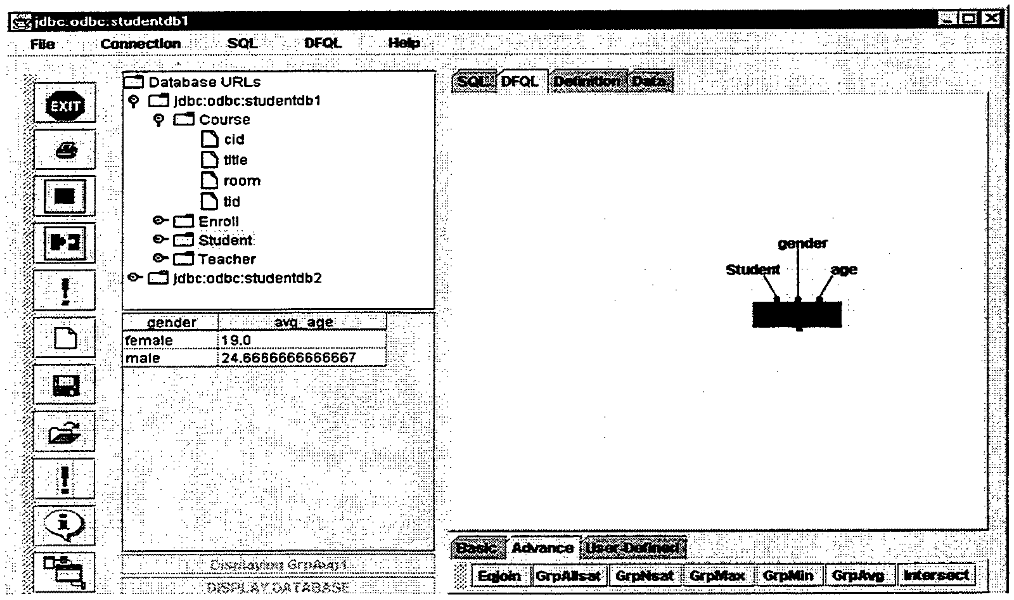
The user selects the *groupAvg* operator from this list, or from the operator tab. After selection, the property window of this operator is displayed. The user gives this operator a name, enters a relation, a grouping attrribute and an aggragate attribute, and chooses a display mode. Then the user clicks the *OK* button to accept the input. Finally, from the *toolbar* or the *DFQL* menu, the user selects the execute button to run this operator.

For this example query, "**Select distinct gender, avg (age) as avg_age from Student group by gender**" is used.



Name	GrpAvg1
Relation	Student
Group Att.	gender
Agg.Att.	age

Figure 4.27: Operator GroupAvg Property Window



Database URLs
jdbc:odbc:studentdb1
Course
cid
title
room
tid
Enroll
Student
Teacher
jdbc:odbc:studentdb2

gender	avg_age
female	19.0
male	24.6666666666667

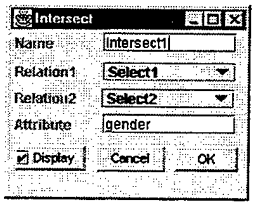
Displaying GrpAvg1
DISPLAY DATABASE

Figure 4.28: Execution of DFQL Operator - GroupAvgn

g. *Execution of DFQL Operator – Intersect*

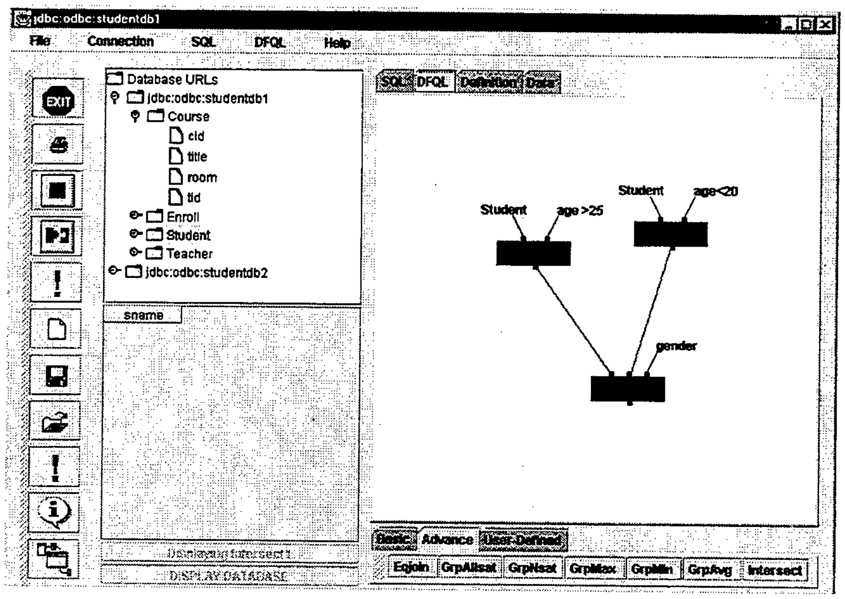
The user selects the *intersect* operator from this list, or from the operator tab. After selection, the property window of this operator is displayed. The user gives this operator a name, enters a relation and an attribute and chooses a display mode. Then the user clicks the *OK* button to accept the input. Finally, from the *toolbar* or the *DFQL* menu, the user selects the execute button to run this operator.

For this example query, "**Select distinct sname from (select distinct * from Student where age >25) S where S.sname in(select sname from (select distinct * from Student where age<20))**" is used.



Name: Intersect1
Relation1: Select1
Relation2: Select2
Attribute: gender
[✓ Display]

Figure 4.29: Operator Intersect Property Window



Displaying Intersect1

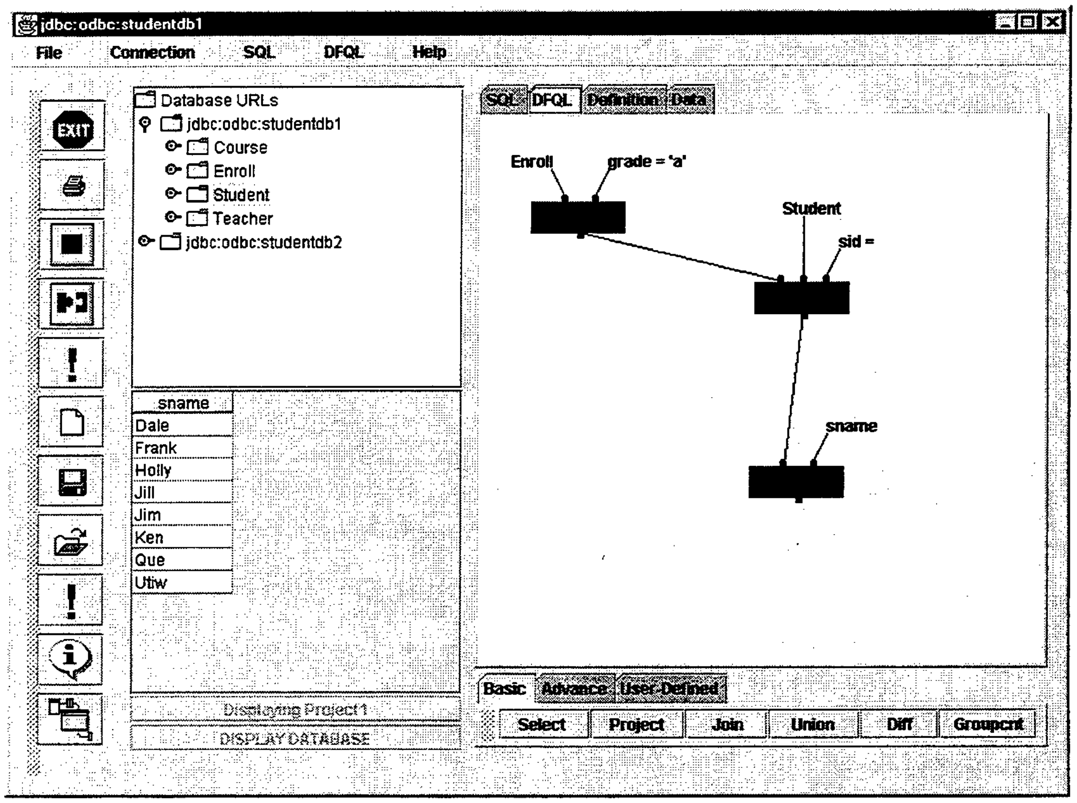
DISPLAY DATABASE

Figure 4.30: Execution of DFQL Operator – Intersect

8. Execution of Incremental Query

The user can construct incremental queries by connecting the operators. Each operator's output is a relation by itself. The query "**select distinct sname from (select distinct * from (select distinct * from Enroll where grade = 'a') j4,Student j41 where j4.sid = j41.sid)**" will be used to show how to create an incremental query in DFQL using the operators. This query retrieves the name of the students who gets a grade of 'A' in at least one of his courses.

First, the query chooses all the records from the "**enroll**" table with condition "**grade = 'a'**". The query joins this result with another relation, "**student**", with the condition that both "**sid**"'s are the same. Finally, program displays the name of the students from the result of the join.

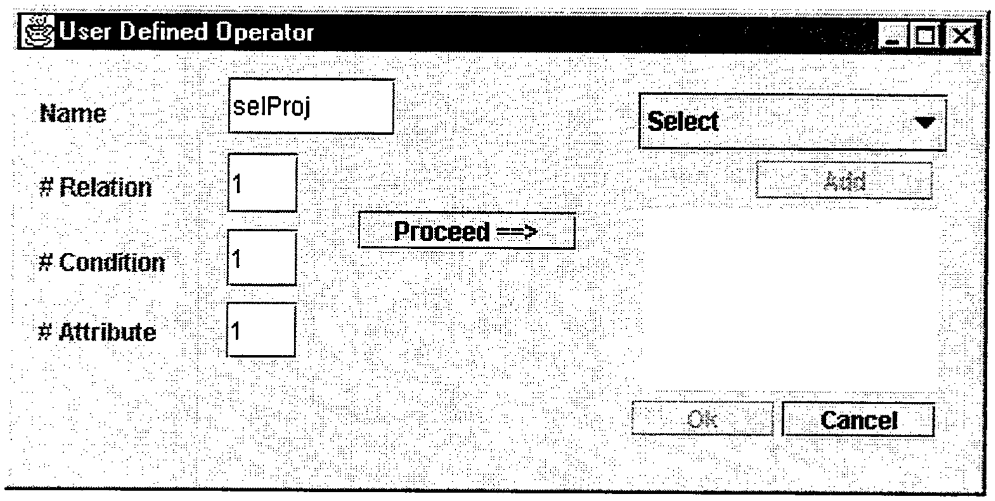


sname
Dale
Frank
Holly
Jill
Jim
Ken
Que
Utiw

Figure 4.31: Execution of an Incremental DFQL Query

9. Execution of User Defined Operator

The user needs to select the main panel's DFQL tab to create a DFQL operator. In order to create a user defined operator, the *user defined panel* must be selected from the *operator panel.* After these selections, clicking on the empty canvas or pressing the *new button* displays the user defined operator construction screen.

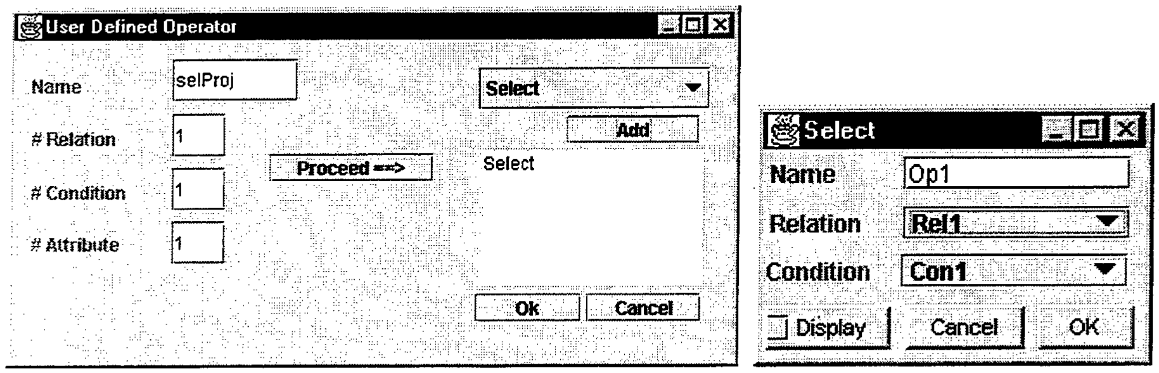


Name: selProj
Relation: 1
Condition: 1
Attribute: 1

Figure 4.32: User Defined Operator Construction Screen

This screen allows user to specify the new operator. The user needs to enter a name and number of the relations, conditions and attributes for this new operator. After entering this information, the user presses the proceed button. For this example, a selproj operator is created, which is the combination of the select and project operators.

Now, the user needs to select which operators are going to be combined to produce the new operator. The user selects the operators from the combo box by choosing the operator and pressing the add button. First, the user selects the *operator select.*


User Defined Operator

Name: selProj

Relation: 1

Condition: 1

Attribute: 1

Select

Select

Select

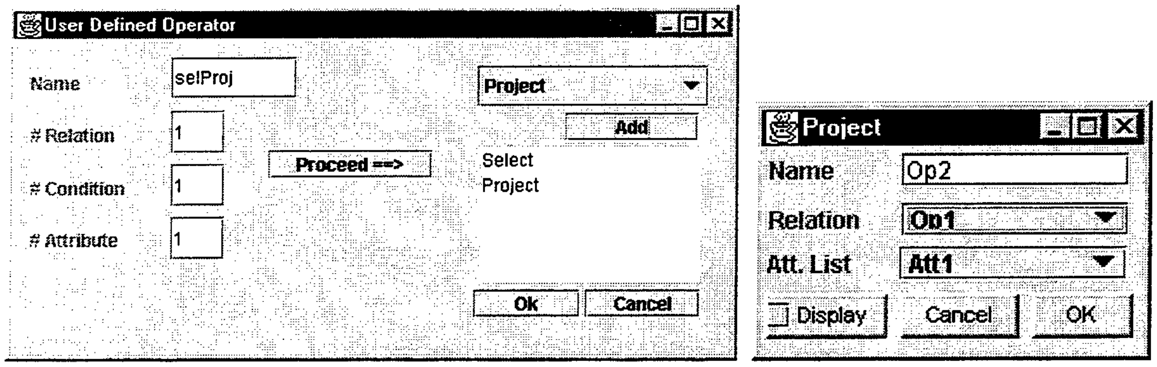
Name: Op1

Relation: Rel1

Condition: Con1

Figure 4.33: Select Operator Configuration

When the user presses the add button, the selected operator's property window pops up. The user selects the proper inputs of the new operator. For this example, the first relation (Rel1) and condition (Con1) inputs of the new operator are connected to the select operator. The user adds a new operator. (Project)


User Defined Operator

Name: selProj

Relation: 1

Condition: 1

Attribute: 1

Project

Select
Project

Project

Name: Op2

Relation: Op1

Att. List: Att1

Figure 4.34: Project Operator Configuration

The second operator is selected and configured, and the output of the first operator (Select) is used as the input of the second operator. At the end of configuration, the user presses the *OK button.* The new operator is automatically saved.

In order to use this new operator, the user needs to load the operator.

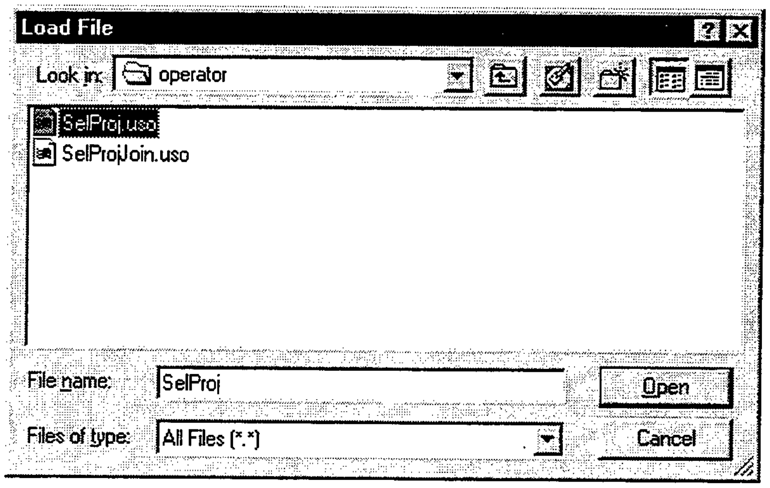


Figure 4.35: Load User Defined Operator

The new operator's property window is displayed, when the loading process ends. The user enters the required information for the new operator. For this example, we used the query, "**select distinct sname from (select distinct * from Student where age>23),**" which retrieves the names of the students who are older than 23.

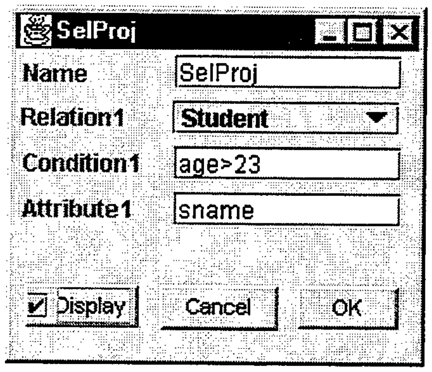
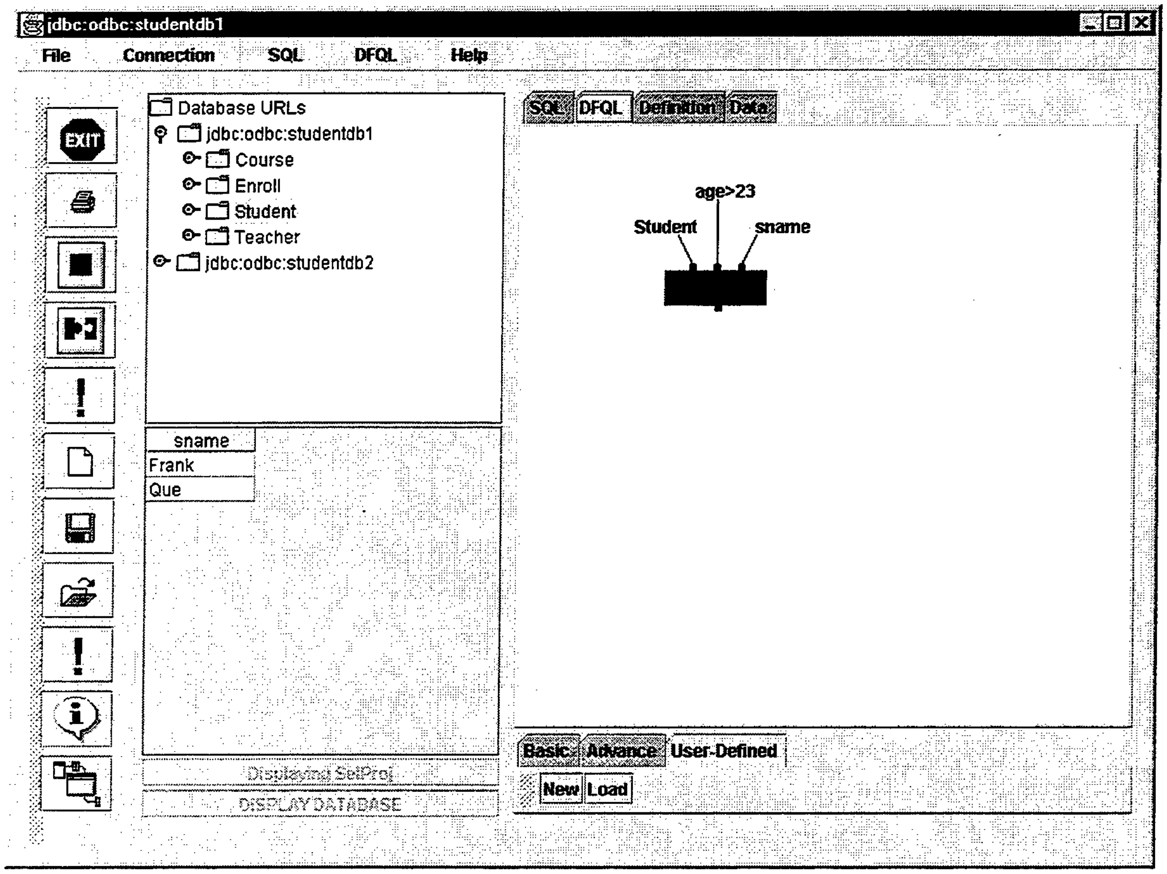


Figure 4.36: SelProj Operator Property Window.

This operator can be placed and run in the same way as the other operators. The user selects the execute button to run this operator.



sname
Frank
Que

Figure 4.37: Run Operator Selproj

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS AND RECOMMENDATIONS

Previous work mentioned in background chapter (Chapter I) establishes that DFQL is capable of solving problems encountered when using SQL and QBE, such as inability to express universal quantification and specify complex nested queries. This thesis research attempts to demonstrate that DFQL can be implemented as powerfully, simply, and intuitivly as its creators stipulate. All of the functionalities of this application have been tested and are working properly based on the design requirements.

This application provides the user with a very simple and powerful graphical user interface. By providing database connection information, the user can log onto one or more databases concurrently.

The application allows user to view the database metadata and the contents of the relations in the database.

Users can query the databases by using either the SQL text editor or the DFQL visual query constructor.

In order to create DFQL queries, the user can use built-in operators, or can create user-defined operators.

Query results can be printed. Queries and user-defined operators can be saved and opened for future use.

Because this is an early version of this application, there are design modifications which should be considered in order to enhance the application's effectiveness.

In our design, *JPanel* class is used to display the DFQL operators. Since the area for the panel cannot be extended, it gives the user a limited area to create operators. Extending the *Scrollable* class is a possible solution for this problem; however this option requires additional coding.

To improve the user friendliness, this application can be designed in a way that, users are able to drag and drop an operator onto the DFQL query design area. On this design area users can click on the input nodes in order to enter the corresponding input values.

This application works with multiple databases running concurrently. Application sends the same query to all databases connected. The problem with is that the databases connected must be identical. They must have the same structure. This feature, working with multiple databases, is designed to allow the user to work with distributed databases.

Although this application was tested several times and the encountered bugs were corrected. Hence it is recommended that a beta version of this application be released. And new patches should be released, according to the user compliances.

APPENDIX – A

SOURCE CODE

1. Advancedoperators.java

```
package ThesisGUI;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

/**
 * This frame is used to display the advanced DFQL operators and allows user to
 * select one of them.
 *
 * AUTHOR : IS & BA
**/

public class AdvancedOperators extends JFrame implements WindowListener {
 /** Operator EQJoin**/
 JButton btnEqjoin    = new JButton();

 /** Operator Group All Satisfy**/
 JButton btnGrpAllSat  = new JButton();

 /** Operator Group Number Satisfy**/
 JButton btnGrpNSat   = new JButton();

 /** Aggregate Operators
  * Maximum**/
 JButton btnGrpMax    = new JButton();

 /** Aggregate Operators
  * Average**/
 JButton btnGrpAvg    = new JButton();

 /** Aggregate Operators
  * Minimum**/
 JButton btnGrpMin    = new JButton();

 /** Operator Intersect**/
 JButton btnIntersect  = new JButton();
```

```
/** Cancel button **/
JButton btnCancel     = new JButton();

/** X coordinate of the new operator**/
int x;

/** Y coordinate of the new operator**/
int y;

/*            CONSTRUCTOR                    */
/**
 * The constructor
 * @param x and y coordinates of the new operator
**/
public AdvancedOperators(int x, int y) {
 try  {
  this.x = x;
  this.y = y;
  jbInit();
  }
 catch(Exception e) {
  e.printStackTrace();
  }
}

/*            INITIATE METHOD                    */
/**
 * The initiation
 * creates the frame, and the buttons. Initiates them.
**/
private void jbInit() throws Exception {
 this.getContentPane().setLayout(null);
 this.setTitle                ("Advanced");
 btnEqjoin.setText            ("Eqjoin");
 btnGrpAllSat.setText         ("GrpAllSat");
 btnGrpNSat.setText           ("GrpNSat");
 btnGrpMax.setText            ("GrpMax");
 btnGrpAvg.setText            ("GrpAvg");
 btnGrpMin.setText            ("GrpMin");
 btnIntersect.setText         ("Intersect");
 btnCancel.setText            ("Cancel");

 btnEqjoin.setBounds         (new Rectangle(1, 3, 111, 16));
 btnEqjoin.addActionListener (new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   btnEqjoin_actionPerformed(e); } });
```

```
btnGrpAllSat.setBounds      (new Rectangle(1, 19, 111, 16));
btnGrpAllSat.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnGrpAllSat_actionPerformed(e);
 }});

btnGrpNSat.setBounds      (new Rectangle(1, 35, 111, 16));
btnGrpNSat.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnGrpNSat_actionPerformed(e);
 }});

btnGrpMax.setBounds      (new Rectangle(1, 51, 111, 16));
btnGrpMax.addActionListener (new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnGrpMax_actionPerformed(e);
 }});

btnGrpAvg.setBounds      (new Rectangle(1, 67, 111, 16));
btnGrpAvg.addActionListener (new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnGrpAvg_actionPerformed(e);
 }});

btnGrpMin.setBounds      (new Rectangle(1, 83, 111, 16));
btnGrpMin.addActionListener (new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnGrpMin_actionPerformed(e);
 }});

btnIntersect.setBounds      (new Rectangle(1, 99, 111, 16));
btnIntersect.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnIntersect_actionPerformed(e);
 } });

btnCancel.setBounds      (new Rectangle(1, 115, 111, 16));
btnCancel.addActionListener (new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnCancel_actionPerformed(e);
 }});

btnCancel.setForeground(Color.red);
```

```
  this.getContentPane().add(btnEqjoin, null);
  this.getContentPane().add(btnGrpNSat, null);
  this.getContentPane().add(btnGrpMax, null);
  this.getContentPane().add(btnGrpAvg, null);
  this.getContentPane().add(btnGrpMin, null);
  this.getContentPane().add(btnGrpAllSat, null);
  this.getContentPane().add(btnIntersect, null);
  this.getContentPane().add(btnCancel, null);

  this.setBounds(x,y,120,160);
  this.show(true);
 }

/*              WINDOW ACTIVATIONS                    */
/**
  * The window listener methods
  * @param event (WindowEvent)
**/
public void windowClosing        (WindowEvent event)  {this.dispose();  }
public void windowActivated      (WindowEvent event){ }
public void windowClosed         (WindowEvent event){ }
public void windowDeactivated    (WindowEvent event){ }
public void windowDeiconified    (WindowEvent event){ }
public void windowIconified      (WindowEvent event){ }
public void windowOpened         (WindowEvent event){ }

/*              ACTION PERFORMED METHODS                    */
/**
  * The Eqjoin button action performed
  * When user selects this button, program creates an
  * EQJoin operator with the given coordinates
 .* @see OperatorEqjoin
**/
void btnEqjoin_actionPerformed(ActionEvent e) {
  OperatorEqjoin eqjoin = new OperatorEqjoin(x,y);
  this.dispose(); }

/**
  * The Group all satisfy button action performed
  * When user selects this button, program creates a
  * group all satisfy operator with the given coordinates
  * @see OperatorGrpAllSat
**/
void btnGrpAllSat_actionPerformed(ActionEvent e) {
  OperatorGrpAllSat grpAllSat = new OperatorGrpAllSat(x,y);
  this.dispose(); }
```

```
/**
  * The Group number satisfy button action performed
  * When user selects this button, program creates a
  * group number satisfy operator with the given coordinates
  * @see OperatorGrpNSat
**/
void btnGrpNSat_actionPerformed(ActionEvent e) {
 OperatorGrpNSat grpNSat = new OperatorGrpNSat(x,y);
 this.dispose();
}

/**
  * The Group maximum button action performed
  * When user selects this button, program creates a
  * group maximum operator with the given coordinates
  * @see OperatorGrpMax
**/
void btnGrpMax_actionPerformed(ActionEvent e) {
 OperatorGrpMax grpMax = new OperatorGrpMax(x,y);
 this.dispose();
}

/**
  * The Group minimum button action performed
  * When user selects this button, program creates a
  * group minimum operator with the given coordinates
  * @see OperatorGrpMin
**/
void btnGrpMin_actionPerformed(ActionEvent e) {
 OperatorGrpMin grpMin = new OperatorGrpMin(x,y);
 this.dispose();
}

/**
  * The Group average button action performed
  * When user selects this button, program creates a
  * group average operator with the given coordinates
  * @see OperatorGrpAvg
**/
void btnGrpAvg_actionPerformed(ActionEvent e) {
 OperatorGrpAvg grpAvg = new OperatorGrpAvg(x,y);
 this.dispose();
}
```

```
  /**
   * The Intersect button action performed
   * When user selects this button, program creates an
   * intersect operator with the given coordinates
   * @see OperatorIntersect
  **/
  void btnIntersect_actionPerformed(ActionEvent e) {
   OperatorIntersect intersect = new OperatorIntersect(x,y);
   this.dispose();
  }

  /**
   * The Cancel button action performed
   * Terminates this frame without creating an operator
  **/
  void btnCancel_actionPerformed(ActionEvent e) {
   this.dispose();
  }
}
```

2. Attribute.Java

```
package ThesisGUI;

import java.util.*;

/**
 * This class holds the data for an attribute.
 * @author IS & BA
**/
Public class Attribute {
  /** Name of the attribute **/
  private String name;

  /** Type of the attribute **/
  private String dataType;

  /** Field size of the attribute **/
  private int fieldSize;

  /** Nullability of the attribute. 0- Not Nullable, 1- Nullable, 2- Unknown**/
  private int nullable;

  /** Check whether the values of this attribute can be used in WHERE clause
     True- Searchable, False - otherwise**/
  private boolean searchable;
```

```
/** Case-sensitivity of the attribute values. True- caseSensitive, False -
   otherwise**/
private boolean caseSensitive;

/*          CONSTRUCTOR                    */
/**
 * The constructor
**/
public Attribute () {
  name              = null;
  dataType          = null;
  fieldSize         = -1;
  nullable          = 2;
  searchable        = true;
  caseSensitive     = false;
}

/*          Get Methods                */
/**
 * Gets the name of the attribute
**/
public String getAttributeName () {
  return name;
}

/**
 * Gets the type of the attribute
**/
public String getAttributeType () {
  return dataType;
}

/**
 * Gets the size of the attribute
**/
public int getAttributeSize () {
  return fieldSize;
}

/**
 * Returns the nullability of the attribute
 * @see nullable
**/
public int isNullable() {
  return nullable;  }
```

```
/**
  * Returns the searchability of the attribute
  * @see searchable
**/
public boolean isSearchable() {
  return searchable;
}

/**
  * Returns the case-sensitivity of the attribute
  * @see caseSensitive
**/
public boolean isCaseSensitive() {
  return caseSensitive;
}

/*              Set Methods                 */
/**
  * Sets the name of the attribute
  * @parameter attName (String)
**/
protected void setAttributeName (String attName) {
  name = attName;
}

/**
  * Sets the type of the attribute
  * @parameter attType (String)
**/
protected void setAttributeType (String attType) {
  dataType = attType;
}

/**
  * Sets the size of the attribute
  * @parameter attSize (String)
**/
protected void setAttributeSize (int attSize) {
  fieldSize = attSize;
}

/**
  * Sets the nullability of the attribute
  * @parameter nullIndex (int)
**/
```

```
  protected void setNullable (int nullIndex){
    nullable = nullIndex;
  }

  /**
   * Sets the searcability of the attribute
   * @parameter search (boolean)
  **/
  protected void setSearchable (boolean search){
    searchable = search;
  }

  /**
   * Sets the case-sensitivity of the attribute
   * @parameter caseSense (boolean)
  **/
  protected void setCaseSensitive (boolean caseSense){
    caseSensitive = caseSense;
  }
}
```

3. BasicOperators.Java

```
package ThesisGUI;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

/**
 * This frame is used to display the basic DFQL operators and allows user to
 * select one of them.
 *
 * AUTHOR : IS & BA
**/
public class BasicOperators extends JFrame implements WindowListener {
 /** Operator Project**/
 JButton btnProject  = new JButton();

 /** Operator Select**/
 JButton btnSelect   = new JButton();

 /** Operator Join**/
 JButton btnJoin     = new JButton();
```

```
/** Operator Union**/
JButton btnUnion    = new JButton();

/** Operator Different**/
JButton btnDiff     = new JButton();

/** Operator Group count**/
JButton btnGroupCnt = new JButton();

/** cancel button **/
JButton btnCancel   = new JButton();

/** X coordinate of the new operator**/
int x;

/** Y coordinate of the new operator**/
int y;

/*              CONSTRUCTOR                    */
/**
 * The constructor
 * @param x and y coordinates of the new operator
**/
public BasicOperators(int x, int y) {
 try  {
  this.x=x;
  this.y=y;
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}

/*              INITIATE METHOD                    */
/**
 * The initiation
 * creates the frame, and the buttons. Initiates them.
**/
private void jbInit() throws Exception {
 this.getContentPane().setLayout(null);
 this.setTitle          ("Basic");
 btnSelect.setText      ("Select");
 btnProject.setText     ("Project");
 btnJoin.setText        ("Join");
```

```
btnUnion.setText    ("Union");
btnDiff.setText     ("Diff");
btnGroupCnt.setText ("GroupCnt");
btnCancel.setText   ("Cancel");

btnSelect.setBounds        (new Rectangle(1, 3, 111, 16));
btnSelect.addActionListener (new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    btnSelect_actionPerformed(e);
  }});

btnProject.setBounds       (new Rectangle(1, 19, 111, 16));
btnProject.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    btnProject_actionPerformed(e);
  }});

btnJoin.setBounds          (new Rectangle(1, 35, 111, 16));
btnJoin.addActionListener (new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    btnJoin_actionPerformed(e);
  }});

btnUnion.setBounds         (new Rectangle(1, 50, 111, 16));
btnUnion.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    btnUnion_actionPerformed(e);
  }});

btnDiff.setBounds          (new Rectangle(1, 66, 111, 16));
btnDiff.addActionListener (new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    btnDiff_actionPerformed(e);
  }});

btnGroupCnt.setBounds      (new Rectangle(1, 82, 111, 16));
btnGroupCnt.addActionListener (new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    btnGroupCnt_actionPerformed(e);
  }});

btnCancel.setBounds        (new Rectangle(1, 102, 111, 16));
btnCancel.addActionListener (new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    btnCancel_actionPerformed(e);
  }});
  btnCancel.setForeground(Color.red);

  this.getContentPane().add(btnSelect, null);
  this.getContentPane().add(btnJoin, null);
  this.getContentPane().add(btnUnion, null);
  this.getContentPane().add(btnDiff, null);
  this.getContentPane().add(btnGroupCnt, null);
  this.getContentPane().add(btnProject, null);
  this.getContentPane().add(btnCancel, null);
  this.setBounds(x,y,120,150);
  this.show(true);
}

/*            WINDOW ACTIVATIONS                */
/**
 * The window listener methods
 * @param event (WindowEvent)
**/
public void windowClosing        (WindowEvent event){this.dispose();}
public void windowActivated      (WindowEvent event){}
public void windowClosed         (WindowEvent event){}
public void windowDeactivated    (WindowEvent event){}
public void windowDeiconified    (WindowEvent event){}
public void windowIconified      (WindowEvent event){}
public void windowOpened         (WindowEvent event){}

/*            ACTION PERFORMED METHODS              */
/**
 * The Select button action performed
 * When user selects this button, program creates an
 * Select operator with the given coordinates
 * @see OperatorSelect
**/
void btnSelect_actionPerformed(ActionEvent e) {
 OperatorSelect select = new OperatorSelect(x,y);
 this.dispose();
}

/**
 * The Project button action performed
 * When user selects this button, program creates an
 * Project operator with the given coordinates
 * @see OperatorProject
**/
void btnProject_actionPerformed(ActionEvent e) {
  OperatorProject project = new OperatorProject(x,y);
  this.dispose();
 }

 /**
  * The Join button action performed
  * When user selects this button, program creates an
  * Join operator with the given coordinates
  * @see OperatorJoin
 **/
 void btnJoin_actionPerformed(ActionEvent e) {
  OperatorJoin join = new OperatorJoin(x,y);
  this.dispose();
 }

 /**
  * The Union button action performed
  * When user selects this button, program creates an
  * Union operator with the given coordinates
  * @see OperatorUnion
 **/
 void btnUnion_actionPerformed(ActionEvent e) {
  OperatorUnion union = new OperatorUnion(x,y);
  this.dispose();
 }

 /**
  * The Diferent button action performed
  * When user selects this button, program creates an
  * Different operator with the given coordinates
  * @see OperatorDiff
 **/
 void btnDiff_actionPerformed(ActionEvent e) {
  OperatorDiff diff = new OperatorDiff(x,y);
  this.dispose();
 }

 /**
  * The Group count button action performed
  * When user selects this button, program creates an
  * Group count operator with the given coordinates
  * @see OperatorGrpCnt
 **/
 void btnGroupCnt_actionPerformed(ActionEvent e) {
  OperatorGroupCnt groupCnt = new OperatorGroupCnt(x,y);
  this.dispose(); }
 /**
  * The Cancel button action performed
  * Terminates this frame without creating an operator
 **/
 void btnCancel_actionPerformed(ActionEvent e) {
  this.dispose();
 }
}
```

4. Database.Java

```
package ThesisGUI;

import java.sql.*;
import javax.swing.*;
import javax.swing.tree.*;
import java.util.*;
import java.util.Date;

/**
 * This class manages the connection with JDBC database. A database object
 * encapsulates the functionalities provided the Connection, Statement, and
 * other JDBC classes
 * @author BA & IS
**/
public class Database {

 /**
  * This object represents the connection session with a database and
  * includes the SQL statements and their results.
 **/
 private Connection connection;

 /**
  * This object is used to send SQL statements to the connected database
 **/
 private Statement statement;

 /**
  * The result set of the last query execution
 **/
 private ResultSet queryResult;
```

```
/**
  * The meta data of the result set from the last query execution
**/
private ResultSetMetaData queryResultMetaData;

/**
  * The current relation being executed
**/
private Table currentQuery;

/**
  * The connection info of the database
  * @see DatabaseInfo
**/
private DatabaseInfo databaseInfo;

/**
  * The meta data of the database
**/
private DatabaseMetaData metaData;

/**
  * If the database supports ANSI92 Full SQL. True-Supports, False-otherwise
**/
private boolean support;

/**
  * The keywords supported by this database
**/
private String keywords;

/**
  * The number of the tables in the database
**/
private int tableCount;

/**
  * Queries executed in the current session
**/
private Vector queries;

/**
  * Boolean to synchronize the program with the database
**/
private boolean askDB;
```

```
/**
 * Boolean to decide if the executed query information is going to be stored.
**/
private boolean store;

/**
 * The time period for program to synchronize with the database (in minutes)
**/
private int updatePeriod;

/**
 * DefaultMutableTreeNode which contains the tree structure of the database
**/
private DefaultMutableTreeNode dbTreeNode;

/*          CONSTRUCTOR              */
/**
 * The constructor
 * @param databaseInfo the information to connect a database
 * @see DatabaseInfo
**/
public Database(DatabaseInfo databaseInfo) {
 this.databaseInfo = databaseInfo;
}

/*          PUBLIC  METHODS              */
/**
 * Connects to a JDBC-compliant database
**/
public void connect () throws ClassNotFoundException, SQLException {
 Class.forName(databaseInfo.driver());
 connection  = DriverManager.getConnection (databaseInfo.url(),
                                            databaseInfo.userid(),
                                            databaseInfo.password());
 statement   = connection.createStatement();
 metaData    = connection.getMetaData();
 support     = metaData.supportsANSI92FullSQL();
 keywords    = metaData.getSQLKeywords();
 tableCount  = 0;
 queries     = new Vector();
 askDB       = false;
 updatePeriod = 15;
 store       = true;
```

```
  // finds and stores the database table information
  initiateTables();
  store      = false;
  constructTreeNode();
 }

/**
  * Disconnects from the connected database
**/
public void disconnect () {
  try {
   statement.close();
   connection.close();
  }
  catch (SQLException e) {
   System.out.println(e.getMessage());
  }
}

/**
 * Executes the query
 * If the same query is executed before, this method returns the relation
 * from the query history vector (queries).
 * @param query is the query to be executed in SQL Syntax
 * @return the table object after the execution of the query
**/
public Table executeQuery (String query) throws SQLException {
 query     = connection.nativeSQL(query);
 // translate query into the native query language of the connected DBMS

 query          = query.toLowerCase();     // convert to lower case.
 Table temp  = checkQuery(query);       // check if the query is executed before

 // if the query is not executed before or it is out of date
 if ( temp == null) {
   queryResult = statement.executeQuery(query);
   if (queryResult != null) {
     queryResultMetaData = queryResult.getMetaData();
     currentQuery = new Table(query,this);
     currentQuery.setTableName("query"+queries.size());
     if (store){
       queries.add(currentQuery);
     }
   }
```

```
      // if the executed query does not produce a result set.
      else {
        currentQuery = null;
      }
    }

    // if the query is executed before
    else {
      compareTimeStamp(temp);

      // If the relation needs to be updated
      if (askDB) {
        // Execute the query
        queryResult = statement.executeQuery(temp.getQuery());
        queryResultMetaData = queryResult.getMetaData();
        // Update the relation data
        temp.updateRelationData();
        askDB = false;
      }
      currentQuery = temp;
    }
    return currentQuery;
  }

  /**
   * @return a vector which contains the table names in the Database
        (String vector)
  **/
  public Vector getTableNames()  {
    Vector tableNames = new Vector();
    for (int nameIndex=0; nameIndex < tableCount; nameIndex++) {
      Table temp = (Table)queries.elementAt(nameIndex);
      tableNames.add(temp.getTableName());
    }
    return tableNames;
  }

  /**
   * @return boolean
   * @see support
  **/
  public boolean isSupportANSI92(){
    return support;
  }
```

```
public String getSQLKeywords(){
  return keywords;
}

/**
 * @return the product name of the database program
**/
public String getDatabaseProgramName () throws SQLException {
  return (metaData.getDatabaseProductName());
}

/**
 * @param tableName the relation name as String
 * @return a vector which contains the attributes of the relation
 * (Attribute Vector)
 * @see Attribute class
**/
public Vector getTableAttributes (String tableName) throws SQLException{
  int tableIndex = findTable(tableName);
  if (tableIndex == -1) {
    return null;
  }
  else {
    Table currentTable = (Table)queries.elementAt(tableIndex);
    return currentTable.getAttributes();
  }
}

/**
 * @param tableName the relation name as String
 * @return a vector which contains the attribute names of the relation
 * (String Vector)
**/
public Vector getTableAttributeNames (String tableName) throws
                                                        SQLException{
  int tableIndex = findTable(tableName);
  if (tableIndex == -1) {
    return null;
  }
  else {
    Table currentTable = (Table)queries.elementAt(tableIndex);
    return currentTable.getAttributeNames();
  }
}
```

```
/**
  * @return DefaultMutableTreeNode - contains the tree model of the database
**/
public DefaultMutableTreeNode getTreeModel () {
  return dbTreeNode;}

/**
  * @param tableName the relation name whose data is being asked
  * @return a vector which contains the table data, provided table name
**/
public Vector getTableData(String tableName) throws SQLException{
  int tableIndex = findTable(tableName);
  if (tableIndex == -1) {
    return null;
  }
  else {
    Table temp = (Table)queries.elementAt(tableIndex);
    executeQuery(temp.getQuery());
    return temp.getData();
  }
}

/**
  * @return all the queries executed in the current session
**/
public Vector getQueryHistory() {
  return queries;
}

/**
  * @return Update time interval
  * @see updatePeriod
**/
public int getUpdateTime(){
  return updatePeriod;
}

public boolean isStore(){
  return store;
}

public boolean isAskDB(){
  return askDB;
}
```

```
/**
  * Sets the 'askDB' data member.
  * This data member is used to tell the program that the relation data needs
  * to be updated. This data member can be set either by the user or
  * by the program itself.
  * @param update the new value of the variable askDB
  * @see askDB
**/
public void setAskDB (boolean update) {
  askDB = update;
}

/**
  * Sets the 'updatePeriod' data member.
  * @param newPeriod the new value of the variable updatePeriod as Integer
  * @see updatePeriod
**/

public void setUpdatePeriod (int newPeriod) {
  updatePeriod = newPeriod;
}

public void setStore (boolean newStore) {
  store = newStore;
}

public String getURL( ) {
  return databaseInfo.url();
}

public String getUserName( ) {
  return databaseInfo.userid();
}

public String getDriver( ) {
  return databaseInfo.driver();
}

/*            PRIVATE  METHODS             */
/**
 * Initiates the relations of the database (including data)
**/
private void initiateTables(){
  try {
    String tableName;
    String [] type = {"TABLE"};
```

```
      ResultSet rs  = metaData.getTables(null,null,null,type);

      while (rs.next()) {
        tableName       = rs.getString(rs.getMetaData().getColumnName(3));
        Table currentTable = this.executeQuery("select * from " + tableName);

        currentTable.setTableName(tableName);
        tableCount++;
      }
    }
    catch (SQLException e ){
      System.out.println(e.getMessage());
    }
  }

  /**
   * @param tableName the relation name (in String)
   * @return the index of the table in the Table vector
  **/
  private int findTable(String tableName) {
    int result = -1;
    tableName = tableName.toLowerCase();
    for (int tableIndex = 0 ; tableIndex < queries.size(); tableIndex++) {
      Table temp = (Table) queries.elementAt(tableIndex);
      if (tableName.equals(temp.getTableName().toLowerCase())) {
        result = tableIndex;
        break;
      }
    }
    return result;
  }

  /**
   * Compares the current date with the query's time stamp. If the difference is
   * greater than the updatePeriod, it sets the askDB parameter to TRUE.
   * @param checkRelation the table object whose time stamp going to be
   *  checked.
   * @see askDB
   * @see updatePeriod
  **/
  private void compareTimeStamp (Table checkRelation) {
    Date executionDate  = checkRelation.getTimeStamp();
    Date currentDate    = new Date();

    int executionMinute = (executionDate.getDay()*24 +
      executionDate.getHours())*60 +executionDate.getMinutes();
```

```
     int currentMinute   = (currentDate.getDay()*24 +
                       currentDate.getHours())*60 +
                       currentDate.getMinutes();

     if (currentMinute-executionMinute >= updatePeriod) {
            askDB = true;
      }
   }

/**
  * Checks if the query is executed before. If so, returns the Table object
  * @param queryCheck the query which will be checked (String)
  * @return a Table object
**/
private Table checkQuery(String queryCheck) {
   Table temp = null;
   for (int index = 0; index < queries.size(); index++) {
      temp = (Table) queries.elementAt(index);
      if (temp.getQuery().equals(queryCheck)){
        break;
      }
      else {
        temp = null;
      }
   }
   return temp;
}

/**
  * Generates the tree model of the database and stores the information
  * in the dbTreeNode data member
  * @see dbTreeNode
**/
private void constructTreeNode(){
   DefaultMutableTreeNode top    = new DefaultMutableTreeNode
                              (databaseInfo.url());
   DefaultMutableTreeNode parent = top;
   DefaultMutableTreeNode node;
   Table             tempT;
   Attribute          tempA;
   Vector             treeAttr;

   // Add the relation names to the tree model
   for (int iter=0; iter < tableCount; iter++) {
     tempT = (Table)queries.elementAt(iter);
     node = new DefaultMutableTreeNode(tempT.getTableName());
```

```
      treeAttr = tempT.getAttributes();

      // Add the attribute names to the appropriate relation
      for (int attIter = 0; attIter < treeAttr.size(); attIter++) {
        tempA = (Attribute)treeAttr.elementAt(attIter);
        node.add(new DefaultMutableTreeNode(tempA.getAttributeName()));
      }
      parent.add(node);
    }
    dbTreeNode = top;
  }

  /*            PROTECTED   METHODS                */

  /**
   * @return the meta data of the last query execution
  **/
  protected ResultSetMetaData getQueryMetaData(){
    return queryResultMetaData;
  }

  /**
   * @return the Result Set of the last query execution
  **/
  protected ResultSet getQueryResultSet() {
    return queryResult;
  }
}
```

5. DatabaseInfo.Java

```
package ThesisGUI;

/**
 * This class keeps the four pieces of data necessary for making a JDBC
 * connection
 *
 * AUTHOR : BA & IS
**/

public class DatabaseInfo {
 /** driver of the database **/
 private String driver;

 /** url of the database **/
 private String url;
```

```
/** user name of the current user **/
private String userid;


/** password for the connection **/
private String password;

/*             CONSTRUCTOR                    */
/**
  * Default constructor
**/
public DatabaseInfo() {

  this.driver          = "sun.jdbc.odbc.JdbcOdbcDriver";
  this.url             = "jdbc:odbc:studentdb";
  this.userid          = "Anonymous";
  this.password        = "guest";
}

/**
  * Constructor with parameters
  * @parameters driver, url, user, pwd (String)
**/
public DatabaseInfo (String driver, String url, String user, String pwd) {
  this.driver   = driver;
  this.url      = url;
  this.userid   = user;
  this.password = pwd ;
}

/*             Get Methods                    */
/**
  * Gets the driver of the database
**/
public String driver() {
  return driver;
}

/**
  * Gets the url of the database
**/
public String url() {
  return url;
}
```

```
 /**
  * Gets the user id of the current user
 **/
 public String userid() {
  return userid;  }

 /**
  * Gets the password for the connection
 **/
 public String password() {
  return password;
 }

 /*              Set Methods                    */
 /**
  * Sets the driver of the database
 **/
 public void setDriver(String driver) {
  this.driver = driver;
 }

 /**
  * Sets the url of the database
 **/
 public void setURL(String url) {
  this.url = url;
 }

 /**
  * Sets the user id of the current user
 **/
 public void setUserid(String user)  {
  this.userid = user;
 }

 /**
  * Sets the password
 **/
 public void setPassword(String pwd) {
  this.password = pwd;
 }
}
```

6. Draw.Java

```
package ThesisGUI;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.border.*;
import java.awt.Toolkit;
import javax.swing.event.*;
import java.util.*;
import com.borland.dbswing.*;
import java.io.*;
import java.awt.Graphics2D;
import com.borland.jbcl.control.*;

/**
  * It draws the operators on the canvas. Has a mouse motion listener
  * according to the mouse actions, draws an operator, deletes it, drags
  * it or displays a frame to change the operator properties.
  * after every action, all canvas repainted.
  *
  * AUTHOR : IS & BA
**/
public class Draw extends JPanel implements MouseMotionListener
{
 /** size of the operators **/
 private   static  int OPERATORLENGTH = 70;

 /** max number of operators **/
 private   static  final int MAXNUMOFOPERATORS = 20;

 /**current number of the operators that has been drawn.**/
 protected int     numOfOperators = 0;

 /** an array holds the drawing information**/
 protected Operator[] operators = new Operator [MAXNUMOFOPERATORS];

 /**if the choosen Operator is occupied (>=0) or empty (-1)**/
 private   int     current = -1;

 /*              CONSTRUCTOR                    */
 /**
   * The constructor
 **/
```

```
  public Draw()
  {
   addMouseListener (new MouseAdapter()
   {
    public void mouseClicked(MouseEvent evt)
    {
     Draw.this.mouseClicked(evt);
    }
   });
   addMouseMotionListener(this);
  }

 /**
  * Automatically called by repaint method,
  * In a for loop, it calls the draw method with the index number of the operator
  * in order to draw all the operators.
 **/
  public void paint(Graphics g)
  {
   for(int i = 0; i<numOfOperators; i++){
    draw(g,i);
   }
   test.window.panelMain.repaint();
  }


 /**
  * Returns the index value of the rectangle occupies this spot
  * if the spot is empty returns -1
 **/
  public int find (int x, int y)
  {
   for (int i = 0; i<numOfOperators;i++){
    if(operators[i].x  <= x && x <= operators[i].x+ OPERATORLENGTH
      && operators[i].y <= y && y <= operators[i].y + OPERATORLENGTH/3 )
         { return i;}
   }
   return -1;
  }

 /**
  * draws the operator ;
  * draws a rectangle,  adds input and output nodes, draws the lines from the
  * input nodes, writes the name of the operator and input information
 **/
```

```
public void draw (Graphics g, int i)
{
 /** Length of the name of the operator in order to center it**/
 int lengthOfName = 6 * operators[i].name.length();

/** Empty space number, which will leave before and after the name**/
 int emptySpace;

 /** The distance between the input nodes, in order to place them
       simetrically**/
 int distanceBtwnNodes;
 /** A boolean value is used to determine whether the operator is connected to
   * another operator or connected to a string**/
 boolean relatedToOtherOperator;

 /** List of the relations of an operator has with the other operators**/
 Vector relationHolder = new Vector();

 /** In order to hold the index of the operator and index of the input node**/
 int [] include = {-1,-1};

 if (lengthOfName<OPERATORLENGTH) {
   emptySpace = (int)(OPERATORLENGTH - lengthOfName)/2;
 }
 else{
   emptySpace = 2;
 }

 distanceBtwnNodes = (int)OPERATORLENGTH/
                          (operators[i].numberOfInputNodes+1);
 relationHolder    = searchForRelation(operators[i].input,i);
 g.drawRect(operators[i].x, operators[i].y, OPERATORLENGTH,
              OPERATORLENGTH/3);

 //operator color
 if (operators[i].display){
    g.setColor(Color.red);
 }
 else{
    g.setColor(Color.gray);
 }

 g.fillRect(operators[i].x, operators[i].y, OPERATORLENGTH,
              OPERATORLENGTH/3);
```

```
  //operator name
  g.setColor(Color.black);
  g.drawString(operators[i].name, operators[i].x+emptySpace, operators[i].y+
                    OPERATORLENGTH/5);

  //node input
  for(int xi = 0; xi<operators[i].numberOfInputNodes; xi++){
   String [] tempString = operators[i].getInput();

   if ((!tempString[xi].equals("none"))&&(!tempString[xi].equals(""))){
    relatedToOtherOperator = true;

    for (int si = 0; si<relationHolder.size();si++)
    {
     include = (int []) relationHolder.elementAt(si);
     if (xi == include[0]){
      line(g,xi,i,distanceBtwnNodes,include);
      relatedToOtherOperator = false;
     }
    }

    if (relatedToOtherOperator)
     string(g,xi,i,distanceBtwnNodes);
    }
   }

 //outputNode
 g.fillRoundRect(operators[i].x+OPERATORLENGTH/2,
           operators[i].y+(int)(OPERATORLENGTH /3),5,5,5,5);
}

/**
 * draws a line from node to other operator or to string
**/
void line (Graphics g,int xi, int i, int distanceBtwnNodes,int [] include){
  int xcoord = operators[i].x+(distanceBtwnNodes*(xi+1)+2);

  g.fillRoundRect(operators[i].x+(distanceBtwnNodes*(xi+1)),
           (operators[i].y - 5),5,5,5,5);

  g.drawLine(xcoord+1,operators[i].y-1,
         operators[include[1]].x+1+OPERATORLENGTH/2,
         operators[include[1]].y+(int)(OPERATORLENGTH/3));
}
```

```
/**
 * writes the name of the relation
**/
void string (Graphics g, int xi, int i,int distanceBtwnNodes){
  int xcoord = operators[i].x+(distanceBtwnNodes*(xi+1)+2);
  int sumx=0;
  float sumy=1;

  g.fillRoundRect(operators[i].x+(distanceBtwnNodes*(xi+1)),
            (operators[i].y - 5),5,5,5,5);

  if (operators[i].numberOfInputNodes==4){
   switch (xi){
    case 0:
     sumx = -2;
     g.drawString(operators[i].input[xi], (xcoord-40),(operators[i].y - 26));
    break;

    case 1:
     sumx = -1;
     sumy = 2;
     g.drawString(operators[i].input[xi], (xcoord-25),(operators[i].y - 51));
    break;

    case 2:
     sumx = 1;
     sumy = 1.5f;
     g.drawString(operators[i].input[xi], (xcoord-5),(operators[i].y - 38));
    break;

    case 3:
     sumx = 2;
     g.drawString(operators[i].input[xi], (xcoord+10),(operators[i].y - 26));
    break;
   }
  }

  else if (operators[i].numberOfInputNodes==3){
   switch (xi){
    case 0:
     sumx = -1;
     g.drawString(operators[i].input[xi], (xcoord-40),(operators[i].y - 26));
    break;
```

```
        case 1:
         sumx = 0;
         sumy = 2;
         g.drawString(operators[i].input[xi], (xcoord-15),(operators[i].y - 51));
        break;

        case 2:
         sumx = 1;
         g.drawString(operators[i].input[xi], (xcoord+10),(operators[i].y - 26));
        break;
      }
    }

    else if (operators[i].numberOfInputNodes==2){
     switch (xi){
      case 0:
       sumx = -1;
       g.drawString(operators[i].input[xi], (xcoord-40),(operators[i].y - 26));
      break;
      case 1:
       sumx = 1;
       g.drawString(operators[i].input[xi], (xcoord+10),(operators[i].y - 26));
      break;
     }
    }

  g.drawLine(xcoord,(operators[i].y - 5),
         xcoord+(sumx*10),(operators[i].y - (int)(sumy*25)));
}

/**
 * creates a new operator and adds it to the array
**/
public void add(int x, int y, String name, int number,
          String [] input, boolean display, String type, String query){

 if(numOfOperators<MAXNUMOFOPERATORS){
   operators[numOfOperators] = new Operator (x,y,name,number, input,
                                       display, type, query);
   test.window.DFQLQuery.addElement(operators[numOfOperators]);
   numOfOperators++;
   repaint();
 }
}
```

```
/**
  * When user double clicks on an operator this method is called and it removes
  * the operator from the operator list.
**/
public void remove(int n){
 numOfOperators--;
 operators[n] = operators[numOfOperators];
 if(current == n)
   current = -1;
 repaint();
}

/**
  * main part senses all mouse actions according to the actions
  * conducts different kind of operations:
  * on an empty place;
  * one right click creates a new operator
  * on an operator;
  * one left click, displays the operator information, allows user to change it
  * two right click, deletes the operator
**/
public void mouseClicked(MouseEvent evt){
 int x = evt.getX();
 int y = evt.getY();
 int xBorders = x + OPERATORLENGTH;
 int yBorders = y + OPERATORLENGTH/3;
 int clickCount;
 int number;
 Operator temp;

 current = find(x,y);

 if(!evt.isMetaDown()){
    clickCount = evt.getClickCount();

    if (current<0){
      if(test.window.panelOperators.getSelectedIndex()==0){
        BasicOperators l = new BasicOperators(x,y);
      }

      else if(test.window.panelOperators.getSelectedIndex()==1){
        AdvancedOperators l = new AdvancedOperators(x,y);
      }
      else if(test.window.panelOperators.getSelectedIndex()==2){
        test.window.designRelation   = new Vector();
        test.window.designCondition = new Vector();
        test.window.designAttribute  = new Vector();
        test.window.designOps          = new Vector();
        test.window.designQueries    = new Vector();
        test.window.setDesign (true);
        UserOperator l = new UserOperator(x,y);
      }
    }

    else if (evt.getClickCount() >= 2){
      for(int bi = 0;bi<test.window.DFQLQuery.size();bi++){
        temp = (Operator)test.window.DFQLQuery.elementAt(bi);

        if (operators[current].name.equals(temp.name)){
          test.window.DFQLQuery.removeElementAt(bi);
        }
      }
      remove (current);
    }
  }
  else{
   if (current>=0){
    if (operators[current].type.equals("Select")){
      number= 0;
      for(int zi = 0;zi<test.window.DFQLQuery.size();zi++){
        temp = (Operator)test.window.DFQLQuery.elementAt(zi);
        if (operators[current].name.equals(temp.name)){
          number = zi;
        }
      }
      temp = (Operator)test.window.DFQLQuery.elementAt(number);

      OperatorSelect select = new OperatorSelect(temp.x,temp.y,current,
                                        number,temp.name,
                                        temp.input[0],
                                        temp.input[1],
                                        temp.display);
    }

    if (operators[current].type.equals("Project")){
      number= 0;
      for(int zi = 0;zi<test.window.DFQLQuery.size();zi++){
        temp = (Operator)test.window.DFQLQuery.elementAt(zi);
        if (operators[current].name.equals(temp.name)){
```

```
      number = zi;
     }
    }
    temp = (Operator)test.window.DFQLQuery.elementAt(number);

    OperatorProject project = new OperatorProject(temp.x,temp.y,current,
                                        number,temp.name,
                                        temp.input[0],
                                        temp.input[1],
                                        temp.display);
  }

  if (operators[current].type.equals("Join")){
    number= 0;
    for(int zi = 0;zi<test.window.DFQLQuery.size();zi++){
      temp = (Operator)test.window.DFQLQuery.elementAt(zi);
      if (operators[current].name.equals(temp.name)){
        number = zi;
      }
    }
    temp = (Operator)test.window.DFQLQuery.elementAt(number);

    OperatorJoin join = new OperatorJoin(temp.x,temp.y,current,number,
                                temp.name,temp.input[0],
                                temp.input[1], temp.input[2],
                                temp.display);
  }

  if (operators[current].type.equals("Union")){
    number= 0;
    for(int zi = 0;zi<test.window.DFQLQuery.size();zi++){
      temp = (Operator)test.window.DFQLQuery.elementAt(zi);
      if (operators[current].name.equals(temp.name)){
        number = zi;
      }
    }
    temp = (Operator)test.window.DFQLQuery.elementAt(number);

    OperatorUnion union = new OperatorUnion(temp.x,temp.y,current,number,
                                  temp.name,temp.input[0],
                                  temp.input[1], temp.display);
  }

  if (operators[current].type.equals("Diff")){
    number= 0;
    for(int zi = 0;zi<test.window.DFQLQuery.size();zi++){
      temp = (Operator)test.window.DFQLQuery.elementAt(zi);
      if (operators[current].name.equals(temp.name)){
        number = zi;
      }
    }
    temp = (Operator)test.window.DFQLQuery.elementAt(number);

    OperatorDiff diff = new OperatorDiff(temp.x,temp.y,current,number,
                                temp.name,temp.input[0],
                                temp.input[1], temp.input[2],
                                temp.display);
  }

  if (operators[current].type.equals("GroupCnt")){
    number= 0;
    for(int zi = 0;zi<test.window.DFQLQuery.size();zi++){
      temp = (Operator)test.window.DFQLQuery.elementAt(zi);
      if (operators[current].name.equals(temp.name)){
        number = zi;
      }
    }
    temp = (Operator)test.window.DFQLQuery.elementAt(number);

    OperatorGroupCnt groupCnt = new OperatorGroupCnt
                                   (temp.x,temp.y,current,
                                   number,temp.name,
                                   temp.input[0],
                                   temp.input[1],
                                   temp.input[2],
                                   temp.display);
  }

  if (operators[current].type.equals("Eqjoin")){
    number= 0;
    for(int zi = 0;zi<test.window.DFQLQuery.size();zi++){
      temp = (Operator)test.window.DFQLQuery.elementAt(zi);
      if (operators[current].name.equals(temp.name)){
        number = zi;
      }
    }
    temp = (Operator)test.window.DFQLQuery.elementAt(number);

    OperatorEqjoin Eqjoin = new OperatorEqjoin
                                (temp.x,temp.y,current,number,
                                temp.name,temp.input[0],
                                temp.input[1],temp.input[2],
                                      temp.display);
}

if (operators[current].type.equals("GrpAllSat")){
 number= 0;
 for(int zi = 0;zi<test.window.DFQLQuery.size();zi++){
  temp = (Operator)test.window.DFQLQuery.elementAt(zi);
  if (operators[current].name.equals(temp.name)){
   number = zi;
  }
 }
 temp = (Operator)test.window.DFQLQuery.elementAt(number);

 OperatorGrpAllSat grpAllSat = new OperatorGrpAllSat(temp.x,temp.y,
                                       current,number,
                                       temp.name,
                                       temp.input[0],
                                       temp.input[1],
                                       temp.input[2],
                                       temp.display);
}

if (operators[current].type.equals("GrpNSat")){
 number= 0;
 for(int zi = 0;zi<test.window.DFQLQuery.size();zi++){
  temp = (Operator)test.window.DFQLQuery.elementAt(zi);
  if (operators[current].name.equals(temp.name)){
   number = zi;
  }
 }
 temp = (Operator)test.window.DFQLQuery.elementAt(number);

 OperatorGrpNSat grpNSat = new OperatorGrpNSat(temp.x,temp.y,current,
                                    number,temp.name,
                                    temp.input[0],
                                    temp.input[1],
                                    temp.input[2],
                                    temp.input[3],
                                    temp.display);
}

if (operators[current].type.equals("GrpMax")){
 number= 0;
 for(int zi = 0;zi<test.window.DFQLQuery.size();zi++){
  temp = (Operator)test.window.DFQLQuery.elementAt(zi);
  if (operators[current].name.equals(temp.name)){
      number = zi;
     }
    }
    temp = (Operator)test.window.DFQLQuery.elementAt(number);

    OperatorGrpMax grpMax = new OperatorGrpMax(temp.x,temp.y,current,
                                    number,temp.name,
                                    temp.input[0],
                                    temp.input[1],
                                    temp.input[2],
                                    temp.display);
  }

  if (operators[current].type.equals("GrpMin")){
    number= 0;
    for(int zi = 0;zi<test.window.DFQLQuery.size();zi++){
     temp = (Operator)test.window.DFQLQuery.elementAt(zi);
     if (operators[current].name.equals(temp.name)){
      number = zi;
     }
    }
    temp = (Operator)test.window.DFQLQuery.elementAt(number);

    OperatorGrpMin grpMin = new OperatorGrpMin(temp.x,temp.y,current,
                                    number,temp.name,
                                    temp.input[0],
                                    temp.input[1],
                                    temp.input[2],
                                    temp.display);
  }

  if (operators[current].type.equals("GrpAvg")){
    number= 0;
    for(int zi = 0;zi<test.window.DFQLQuery.size();zi++){
     temp = (Operator)test.window.DFQLQuery.elementAt(zi);
     if (operators[current].name.equals(temp.name)){
      number = zi;
     }
    }
    temp = (Operator)test.window.DFQLQuery.elementAt(number);

    OperatorGrpAvg grpAvg = new OperatorGrpAvg(temp.x,temp.y,current,
                                    number,temp.name,
                                    temp.input[0],
                                    temp.input[1],
                                    temp.input[2],
                                          temp.display);
    }

    if (operators[current].type.equals("Intersect")){
     number= 0;
     for(int zi = 0;zi<test.window.DFQLQuery.size();zi++){
      temp = (Operator)test.window.DFQLQuery.elementAt(zi);
      if (operators[current].name.equals(temp.name)){
       number = zi;
      }
     }
     temp = (Operator)test.window.DFQLQuery.elementAt(number);

     OperatorIntersect intersect = new OperatorIntersect(temp.x,temp.y,
                                          current,number,
                                          temp.name,
                                          temp.input[0],
                                          temp.input[1],
                                          temp.input[2],
                                          temp.display);
    }
   }
  }
 }

/**
 * follows mouse movements
**/
public void mouseMoved(MouseEvent evt){
 int x = evt.getX();
 int y = evt.getY();

 if (find(x,y) >= 0)
  setCursor(Cursor.getPredefinedCursor(1));
 else {
  setCursor(Cursor.getDefaultCursor());
  current = -1;
 }
}

/**
 * In order to move the operators.
 * By pressing the right mouse button and dragging the mouse user can move
 * the operator and replace it.
**/
public void mouseDragged(MouseEvent evt){
  int x = evt.getX();
  int y = evt.getY();

  if (current < 0) {
   current = find (x,y);
  }
  if (current >= 0){
   Graphics g = getGraphics();
   g.setXORMode(getBackground());
   operators[current].x = x;
   operators[current].y = y;
   g.dispose();
   repaint();
  }
 }

/**
  * deletes all the operators
**/
public void clear() {
 for(int xi = 0; xi<numOfOperators;xi++){
   operators[xi] = null;
 }
 numOfOperators = 0;
 current = -1;
 repaint();
}

/**
 * This method looks for a relation between the operators.
**/
public Vector searchForRelation(String [] input, int i){
 Vector relationList = new Vector();
 for(int xi=0;xi<numOfOperators;xi++){
  for(int yi = 0; yi<operators[i].numberOfInputNodes;yi++){
    if(input[yi].equals(operators[xi].name)){
     int [] result = {yi,xi};
     relationList.addElement(result);
    }
  }
 }
 return relationList;
}
```

```
/**
 * calls the current operator and makes the required changes on it.
**/
public void change(int index, int indNumber, int x, int y, String name,
                int number, String [] input, boolean display, String type,
                String query){
  operators[index].setX(x);
  operators[index].setY(y);
  operators[index].setName(name);
  operators[index].setInput(input);
  operators[index].setNumberOfInputNodes(number);
  operators[index].setDisplay(display);
  operators[index].setType(type);
  operators[index].setQuery(query);
  ((Operator)test.window.DFQLQuery.elementAt(indNumber)).setX(x);
  ((Operator)test.window.DFQLQuery.elementAt(indNumber)).setY(y);


((Operator)test.window.DFQLQuery.elementAt(indNumber)).setName(name);
  ((Operator)test.window.DFQLQuery.elementAt(indNumber)).setInput(input);

((Operator)test.window.DFQLQuery.elementAt(indNumber)).setNumberOfInputNodes
                                                         (number);

((Operator)test.window.DFQLQuery.elementAt(indNumber)).setDisplay(display);
  ((Operator)test.window.DFQLQuery.elementAt(indNumber)).setType(type);

((Operator)test.window.DFQLQuery.elementAt(indNumber)).setQuery(query);
  repaint();
 }
}
```

7. Help.Java

```
package ThesisGUI;

import java.awt.event.*;
import java.io.*;
import java.util.*;
import javax.swing.*;
import java.awt.*;
import javax.swing.JFrame;
import java.net.*;
import javax.swing.event.*;
import java.lang.*;
import javax.swing.text.*;
```

```
/**
 * Displays the user manual
 *
 * AUTHOR : IS & BA
**/

public class Help extends JFrame implements HyperlinkListener{
 JEditorPane deneme = new JEditorPane();
 JScrollPane dene   = new JScrollPane(deneme);

 public Help() {
  try  {
   jbInit();
  }
  catch(Exception e) {
   System.out.println(e.getMessage());
  }
 }

 private void jbInit() throws Exception {
  this.setTitle("User Manual for DFQL application");
  deneme.setEditable(false);
  this.getContentPane().add(dene,null);
  setBounds(100,100,400,300);

  try {
    deneme.setPage
        ("file:///"+test.window.getProgramPath()+"/manual/index.html");
    deneme.setCursor
        (Cursor.getPredefinedCursor(Cursor.DEFAULT_CURSOR));
    deneme.addHyperlinkListener(this);
  }
  catch (MalformedURLException ed){
   System.out.println(ed.getMessage());
  }
  catch (IOException e){
   System.out.println(e.getMessage());
  }
  show(true);
 }

 public void hyperlinkUpdate( HyperlinkEvent event ){
        if( event.getEventType() == HyperlinkEvent.EventType.ACTIVATED ){
                // Load some cursors
                Cursor cursor = deneme.getCursor();
                Cursor waitCursor = Cursor.getPredefinedCursor
                                          (Cursor.WAIT_CURSOR );
                deneme.setCursor(waitCursor );

                // Handle the hyperlink change
                SwingUtilities.invokeLater( new PageLoader(deneme,
                                               event.getURL(), cursor ) );
                }
        else if ( event.getEventType() == HyperlinkEvent.EventType.
                                                      ENTERED ){
                // Load some cursors
                 Cursor linkCursor = Cursor.getPredefinedCursor(
                                          Cursor.HAND_CURSOR );
                 deneme.setCursor(linkCursor );
                }

        else if( event.getEventType() == HyperlinkEvent.EventType.EXITED ){
                // Load some cursors
                Cursor defCursor = Cursor.getPredefinedCursor(
                                          Cursor.DEFAULT_CURSOR );
                deneme.setCursor(defCursor );
        }
}

class PageLoader implements Runnable {
 private JEditorPane html;
 private URL        url;
 private Cursor     cursor;

 PageLoader( JEditorPane html, URL url, Cursor cursor ){
   this.html = html;
   this.url = url;
   this.cursor = cursor;
 }

 public void run(){
    if( url == null ){
      // restore the original cursor
      html.setCursor( cursor );
    }
   else {
      Document doc = html.getDocument();
      try {
           html.setPage( url );
      }
      catch( IOException ioe ){
```

```
                html.setDocument( doc );
            }
            catch (NullPointerException deded){
                    System.out.println(deded.getMessage());
            }
            finally {
               // schedule the cursor to revert after
               // the paint has happended.
                  url = null;
                SwingUtilities.invokeLater( this );
             }
          }
       }
     }
   }
```

8. Logon.Java

```
package ThesisGUI;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.sql.*;
import java.io.*;
import java.util.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;

/**
 * This frame is used to connect to a database. It allows user to enter
 * the required information (url, driver, username, password) and according to
 * the information entered, it creates a database object. When this class first
 * called default values for a local student database are displayed.
 *
 * AUTHOR : IS & BA
**/

public class Logon extends JFrame {

 /** Label user name**/
 JLabel lblUserName      = new JLabel();

 /** Label password**/
 JLabel lblPassword      = new JLabel();
```

```
/** Label url**/
JLabel lblURL          = new JLabel();

/** Label driver**/
JLabel lblDriver       = new JLabel();

/** Text field user name, where the user enters his/her name**/
JTextField txtUserName  = new JTextField();

/** Text field password, where the user enters his/her password**/
JTextField txtPassword  = new JTextField();

/** Text field user name, where the user enters database's url**/
JTextField txtURL       = new JTextField();

/** Text field user name, where the user enters database's driver**/
JTextField txtDriver    = new JTextField();

/** Cancel button**/
JButton btnCancel       = new JButton();

/** Okay button**/
JButton btnOK           = new JButton();

/** A database object encapsulates the functionalities provided the
  * Connection, Statement, and other JDBC classes
  * @see Database
  **/
Database database;

/**
  * The connection info of the database
  * @see DatabaseInfo
**/
DatabaseInfo databaseInfo;

/*              CONSTRUCTOR                    */
/**
  * The constructor
**/
public Logon() {
 try {
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}

/*              INITIATE METHOD                    */
/**
 * The initiation
 * creates the frame, and the buttons. Initiates them.
**/
private void jbInit() throws Exception {

 this.setTitle               ("Enter connection information");
 lblUserName.setText         ("User Name :");
 btnCancel.setText           ("Cancel");
 lblPassword.setText         ("Password :");
 lblURL.setText              ("Database URL :");
 lblDriver.setText           ("Driver :");
 btnOK.setText               ("OK");
 txtUserName.setText         ("anonymous");
 txtPassword.setText         ("guest");
 txtURL.setText              ("jdbc:odbc:studentdb1");
 txtDriver.setText           ("sun.jdbc.odbc.JdbcOdbcDriver");

 lblUserName.setBounds       (new Rectangle(8, 16, 106, 19));
 lblPassword.setBounds       (new Rectangle(8, 41, 106, 19));
 lblURL.setBounds            (new Rectangle(8, 65, 106, 19));
 lblDriver.setBounds         (new Rectangle(8, 90, 106, 19));
 txtUserName.setBounds       (new Rectangle(113, 15, 196, 21));
 txtPassword.setBounds       (new Rectangle(113, 40, 196, 21));
 txtURL.setBounds            (new Rectangle(113, 64, 196, 21));
 txtDriver.setBounds         (new Rectangle(113, 89, 196, 21));
 btnCancel.setBounds         (new Rectangle(8, 119, 146, 18));
 btnOK.setBounds             (new Rectangle(163, 119, 146, 18));

 btnCancel.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   btnCancel_actionPerformed(e);
  }});

 btnOK.addActionListener (new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   btnOK_actionPerformed(e);
  }});

 this.getContentPane().setLayout(null);
 this.getContentPane().add(lblUserName, null);
```

```
  this.getContentPane().add(txtUserName, null);
  this.getContentPane().add(lblPassword, null);
  this.getContentPane().add(lblURL, null);
  this.getContentPane().add(lblDriver, null);
  this.getContentPane().add(txtPassword, null);
  this.getContentPane().add(txtURL, null);
  this.getContentPane().add(btnOK, null);
  this.getContentPane().add(btnCancel, null);
  this.getContentPane().add(txtDriver, null);
  setBounds(350,200,330,180);
  show(true);
}

/**
  * The Cancel button action performed
  * Terminates this frame without connecting to a database
**/
void btnCancel_actionPerformed(ActionEvent e) {
  this.dispose();
}

/**
  * The Okay button action performed
  * When this button is pressed, it creates a databaseinfo object,
  * sets its values with the entered information,
  * according to the database info creates a database object,
  * connects to this database,
  * adds this database object to the active database list,
  * modifies the tree,
  * gives a connected message to the user
  * if the connection can not be established gives an error message
**/
void btnOK_actionPerformed(ActionEvent e) {

 databaseInfo = new DatabaseInfo();
 databaseInfo.setUserid  (txtUserName.getText());
 databaseInfo.setPassword(txtPassword.getText());
 databaseInfo.setDriver  (txtDriver.getText());
 databaseInfo.setURL     (txtURL.getText());
 try{
   database = new Database (databaseInfo);
   database.connect();

   test.window.databaseList.add (database);
   test.window.currentDatabase  = database;
```

```
    test.window.generateTree();
    test.window.setTitle(database.getURL());
    test.window.panelMain.setEnabled(true);

    test.window.message.setMessage("Connected : " + database.getURL());
    test.window.message.show      (true);
   }
   catch (ClassNotFoundException ey){
    database = null;
    test.window.message.setMessage ("not connected... "+ey.getMessage());
    test.window.message.show      (true);
   }
   catch (SQLException ex){
    database = null;
    test.window.message.setMessage ("not connected... "+ex.getMessage());
    test.window.message.show      (true);
   }
   this.dispose();
  }
}
```

9. MainTest.Java

```
package ThesisGUI;

import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.sql.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;
import javax.swing.tree.*;
import javax.swing.event.*;
import java.beans.*;

/**
 * This is the main frame
 * It is a user interface, which allows us to make connections to the databases,
 * see these database informations, write SQL queries or create DFQL queries,
 * execute these queries and display results.
 * It has three different part, first (west) is a scroll panel+text area
 * is used to display the meta data, second (north east) is tabpanel
 * has four different panels, one is text area for SQL statement, one
 * is a canvas for DFQL statement, and two text areas for database information
 * and last one (south east) is a scroll panel+text area is used to display
 * results of queries
 *
 * AUTHOR : BA & IS
**/
public class MainTest extends JFrame implements WindowListener{
 /** Counter for each operator **/
 protected int []      Counter      = {1,1,1,1,1,1,1,1,1,1,1,1,1};

 /** It is the canvas on which we draw our operators **/
 protected Draw        cnvDFQL      = new Draw();

 /** Holds the databases that we connected **/
 protected Vector      databaseList = new Vector();

 /** Displays warning, information, error and exception messages **/
 protected Message     message      = new Message ();

 /** Ensures the relation with the database results and result table **/
 protected Relation    relation     = new Relation();

 /** Used to display the table information when clicked on it **/
 protected Relation    relationTable = new Relation();

 /** Holds the operators that we created **/
 protected Vector      DFQLQuery    = new Vector();

 /** Query that we created **/
 protected String      sqlQuery;

 /** Active database connection **/
 protected Database    currentDatabase;

 /** Displays the user manual **/
 protected Help        help;

 /** Temporary variable database type **/
 protected Database    tempDatabase;

 /** Temporary variable Operator type **/
 protected Operator    tempOperator;

 /** Counter used for displaying each operator in order **/
 protected int         index;
```

```
/** Counter used for connecting to each database in order**/
protected int          indexDatabase;

/** holds the active panel information (SQL, DFQL, Definition, Data) **/
protected int          indexPanel= 0;

/** Height of the main frame **/
private   static final int HEIGHT = 600;

/** Width of the main frame **/
private   static final int WIDTH  = 800;

private final String path = System.getProperty("user.dir");

/** Used to display database information **/
JTree      treeMeta      = new JTree();

/** Main (Tabbed) panel contains 4 tabs **/
JTabbedPane panelMain      = new JTabbedPane();

/** This (Tabbed) panel for operators contains 3 tabs **/
JTabbedPane panelOperators = new JTabbedPane();

/** This panel holds the tree meta, it is used to display database info **/
JScrollPane panelMeta      = new JScrollPane(treeMeta);

/**This panel holds the DFQL query design area**/
JscrollPane denememe     = new JScrollPane ();

/**This table is used to display sql and dfql queries results **/
JTable     SQLResult     = new JTable(relation);

/** This table is used to display database table info **/
JTable     tableData     = new JTable(relationTable);

/** Scroll panel holds the SQLResults table **/
JScrollPane panelResults  = new JScrollPane(SQLResult);

/** Scroll panel holds the tableData table **/
JScrollPane panelTableData = new JScrollPane(tableData);

/** When using DFQL, this button allows user to execute the nest operator and
 * display its results. **/
JButton    btnNext1      = new JButton("NEXT OPERATOR");
```

```
/** When working with multiple databases, this button allows user to change the
  database that the query will be executed **/
JButton    btnNext2      = new JButton("NEXT DATABASE");

/** Text area allows user to enter sql queries **/
JTextArea  txtSQL        = new JTextArea();

/** This text area is called definition area. Displays the details of database
  * and attribute information **/
JTextArea  metaData      = new JTextArea();

/** This image icon is used to place the images of the program **/
ImageIcon  icon;

//toolBasic
/** Toolbar for basic operators **/
JToolBar toolBasic  = new JToolBar();

/** Operator select button **/
JButton btnSelect   = new JButton ("    Select     ");

/** Operator project button **/
JButton btnProject  = new JButton ("    Project    ");

/** Operator join button **/
JButton btnJoin     = new JButton ("     Join      ");

/** Operator union button **/
JButton btnUnion    = new JButton ("    Union      ");

/** Operator different button **/
JButton btnDiff     = new JButton ("     Diff      ");

/** Operator group count button **/
JButton btnGroupCnt = new JButton ("   Groupcnt  ");

//toolAdvance
/** Toolbar for advance operators **/
JToolBar toolAdvance  = new JToolBar();

/** Operator Eqjoin button **/
JButton btnEqjoin     = new JButton (" Eqjoin  ");

/** Operator group all satisfy button **/
JButton btnGrpAllSat  = new JButton ("GrpAllsat ");
```

```
/** Operator group number satisfy button **/
JButton btnGrpNSat    = new JButton (" GrpNsat ");

/** Operator group maximum button **/
JButton btnGrpMax     = new JButton (" GrpMax  ");

/** Operator group minimum button **/
JButton btnGrpMin     = new JButton (" GrpMin  ");

/** Operator group average button **/
JButton btnGrpAvg     = new JButton (" GrpAvg  ");

/** Operator intersect button **/
JButton btnIntersect  = new JButton (" Intersect ");

//toolUserDefined
/** Toolbar for user defined operators **/
JToolBar toolUser1    = new JToolBar();


/** Operator User button **/
JButton userNew     = new JButton ("New");

/** Loads saved user operator button **/
JButton userLoad    = new JButton ("Load");

//toolBar
/** Toolbar for program tools **/
JToolBar toolBar = new JToolBar();

/** Exit button terminates program **/
JButton btnExit;

/** Print button prints the query results **/
JButton btnPrint;

/** Connect button connects to database **/
JButton btnConnect;

/** Disconnect button disconnects from the database **/
JButton btnDisConnect;

/** SQL execute button, executes the SQL query **/
JButton btnSQLExecute;
```

```
/** New button, clears all the windows **/
JButton btnNew;

/** Save button, allows user to save the DFQL operator that is created **/
JButton btnSave;

/** Open button, allows user to load previously saved operator **/
JButton btnOpen;

/** DFQL execute button executes the query produced by dfql operators **/
JButton btnDFQLExecute;

/** Manual button, displays the user manual for the program **/
JButton btnManual;

/** Send all button, executes the query with all of the databases connected **/
JButton btnSendAll;

//Menu
/** Pull down menu for program tools **/
JMenuBar  menuBar;

/** File menu (Exit, print) **/
JMenu     menuFile;

/** Exit menu item **/
JMenuItem menuFileExit;

/** Print menu item **/
JMenuItem menuFilePrint;

/** Connection menu (connect, disconnect) **/
JMenu     menuConnection;

/** Connect menu item **/
JMenuItem menuConnectionConnect;

/** Disconnect menu item **/
JMenuItem menuConnectionDisConnect;

/** SQL menu (SQLExecute) **/
JMenu     menuSQL;

/** SQLExecute menu item **/
JMenuItem menuSQLExecute;
```

```
/** DFQL menu (new, save, open, DFQLExecute) **/
JMenu    menuDFQL;

/** New menu item **/
JMenuItem menuDFQLNew;

/** Save menu item **/
JMenuItem menuDFQLSave;

/** Open menu item **/
JMenuItem menuDFQLOpen;

/** DFQLExecute menu item **/
JMenuItem menuDFQLExecute;

/** Help menu (manual) **/
JMenu    menuHelp;

/** Manual menu item **/
JMenuItem menuHelpManual;
JMenuItem menuSettings = new JMenuItem();

/** User defined operator vectors**/
boolean design;
Vector designRelation;
Vector designCondition;
Vector designAttribute;
Vector designOps;
Vector designQueries;

/*             CONSTRUCTOR                    */
/**
 * The constructor
**/
public MainTest() {
 try {
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}
```

```
/*              INITIATE METHOD                    */
/**
 * The initiation
 * creates the mainframe, panels, buttons, text areas, menu items and
 * tables, and initiates them.
**/
private void jbInit() throws Exception {
 addWindowListener   (this);
 this.setSize        (WIDTH, HEIGHT);
 this.setTitle       ("No Current Database Connection");


//*********************************************************************
        //ToolBar : Program tools
//*********************************************************************

 toolBar.setBounds(new Rectangle(10, 15, (int)(WIDTH/10),
                    (int)(HEIGHT*21/24)));
 toolBar.setLayout(null);
 icon   = new ImageIcon (path+"\\images\\exit.gif","Exit");
 btnExit = new JButton(icon);
 btnExit.setToolTipText("Exit");
 btnExit.setBounds   (new Rectangle (10,10,(int)(WIDTH/16),
                    (int)(HEIGHT/15)));
 btnExit.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   btnExit_actionPerformed(e);
  }});
 btnExit.setHorizontalAlignment  (SwingConstants.CENTER);
 btnExit.setVerticalAlignment    (SwingConstants.TOP);

 icon     = new ImageIcon (path+"\\images\\print.gif","Print");
 btnPrint  = new JButton(icon);
 btnPrint.setToolTipText("Print");
 btnPrint.setBounds(new Rectangle(10,((int)(HEIGHT/15)+15),
                    (int)(WIDTH/16),(int)(HEIGHT/15)));

 btnPrint.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   btnPrint_actionPerformed(e);
  }});
 btnPrint.setHorizontalAlignment (SwingConstants.CENTER);
 btnPrint.setVerticalAlignment   (SwingConstants.CENTER);

 icon      = new ImageIcon (path+"\\images\\connect.gif","Connect");
 btnConnect  = new JButton(icon);
```

```
btnConnect.setToolTipText("Connect");
btnConnect.setBounds(new Rectangle(10,2*(int)(HEIGHT/15)+20,
                        (int)(WIDTH/16), (int)(HEIGHT/15)));
btnConnect.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnConnect_actionPerformed(e);
 }});
btnConnect.setHorizontalAlignment (SwingConstants.CENTER);
btnConnect.setVerticalAlignment   (SwingConstants.CENTER);

icon         = new ImageIcon (path+"\\images\\disconnect.gif","Disconnect");
btnDisConnect = new JButton(icon);
btnDisConnect.setToolTipText("Disconnect");
btnDisConnect.setBounds(new Rectangle(10,3*(int)(HEIGHT/15)+25,
                          (int)(WIDTH/16),(int)(HEIGHT/15)));
btnDisConnect.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnDisConnect_actionPerformed(e);
 }});
btnDisConnect.setHorizontalAlignment  (SwingConstants.CENTER);
btnDisConnect.setVerticalAlignment    (SwingConstants.CENTER);

btnSQLExecute = new JButton("!");
btnSQLExecute.setFont(new java.awt.Font("Dialog", 1, 36));
btnSQLExecute.setForeground(Color.red);
btnSQLExecute.setToolTipText("SQLExecute");
btnSQLExecute.setBounds(new Rectangle(10,4*(int)(HEIGHT/15)+30,
                          (int)(WIDTH/16),(int)(HEIGHT/15)));
btnSQLExecute.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnSQLExecute_actionPerformed(e);
 }});

icon   = new ImageIcon (path+"\\images\\New.gif","New");
btnNew = new JButton(icon);
btnNew.setToolTipText("New");
btnNew.setBounds(new Rectangle(10,5*(int)(HEIGHT/15)+35,
                    (int)(WIDTH/16),(int)(HEIGHT/15)));
btnNew.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnNew_actionPerformed(e);
 }});
btnNew.setHorizontalAlignment (SwingConstants.CENTER);
btnNew.setVerticalAlignment   (SwingConstants.CENTER);

icon   = new ImageIcon (path+"\\images\\Save.gif","Save");
```

```
btnSave = new JButton(icon);
btnSave.setToolTipText("Save");
btnSave.setBounds(new Rectangle(10,7*(int)(HEIGHT/15),
                  (int)(WIDTH/16),(int)(HEIGHT/15)));
btnSave.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnSave_actionPerformed(e);
 }});
btnSave.setHorizontalAlignment  (SwingConstants.CENTER);
btnSave.setVerticalAlignment    (SwingConstants.CENTER);

icon    = new ImageIcon (path+"\\images\\open.gif","Open");
btnOpen = new JButton(icon);
btnOpen.setToolTipText("Open");
btnOpen.setBounds(new Rectangle(10,8*(int)(HEIGHT/15)+5,
                  (int)(WIDTH/16),(int)(HEIGHT/15)));
btnOpen.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnOpen_actionPerformed(e);
 }});
btnOpen.setHorizontalAlignment  (SwingConstants.CENTER);
btnOpen.setVerticalAlignment    (SwingConstants.CENTER);

btnDFQLExecute = new JButton("!");
btnDFQLExecute.setFont(new java.awt.Font("Dialog", 1, 36));
btnDFQLExecute.setForeground(Color.blue);
btnDFQLExecute.setToolTipText("DFQLExecute");
btnDFQLExecute.setBounds(new Rectangle(10,9*(int)(HEIGHT/15)+10,
                         (int)(WIDTH/16),(int)(HEIGHT/15)));
btnDFQLExecute.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnDFQLExecute_actionPerformed(e);
 }});

icon = new ImageIcon (path+"\\images\\About.gif","About");
btnManual = new JButton(icon);
btnManual.setToolTipText("Manual");
btnManual.setBounds(new Rectangle(10,10*(int)(HEIGHT/15)+15,
                    (int)(WIDTH/16),(int)(HEIGHT/15)));
btnManual.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnManual_actionPerformed(e);
 }});
btnManual.setHorizontalAlignment  (SwingConstants.CENTER);
btnManual.setVerticalAlignment    (SwingConstants.CENTER);
```

```
icon = new ImageIcon (path+"\\images\\sendAll.gif","SendAll");
btnSendAll = new JButton(icon);
btnSendAll.setToolTipText("Send to all databases");
btnSendAll.setBounds(new Rectangle(10,11*(int)(HEIGHT/15)+20,
                    (int)(WIDTH/16),(int)(HEIGHT/15)));
btnSendAll.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnSendAll_actionPerformed(e);
 }});

btnSendAll.setHorizontalAlignment (SwingConstants.CENTER);
btnSendAll.setVerticalAlignment   (SwingConstants.CENTER);

txtSQL.addComponentListener(new java.awt.event.ComponentAdapter() {
 public void componentShown(ComponentEvent e) {
  txtSQL_componentShown(e);
 }});

metaData.addComponentListener(new java.awt.event.ComponentAdapter() {
 public void componentShown(ComponentEvent e) {
  metaData_componentShown(e);
 }});

panelTableData.addComponentListener(new
     java.awt.event.ComponentAdapter() {
 public void componentShown(ComponentEvent e) {
  panelTableData_componentShown(e);
 }});

denememe.getViewport().add(cnvDFQL);
denememe.addComponentListener(new java.awt.event.ComponentAdapter() {
 public void componentShown(ComponentEvent e) {
  denememe_componentShown(e);
 }});

txtSQL.setNextFocusableComponent(cnvDFQL);
txtSQL.setLineWrap(true);
menuSettings.setText("Settings...");

menuSettings.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  menuSettings_actionPerformed(e);
 }});
toolBar.add(btnExit);
toolBar.add(btnManual);
toolBar.add(btnSendAll);
```

```
          toolBar.add(btnPrint);
          toolBar.add(btnConnect);
          toolBar.add(btnDisConnect);
          toolBar.add(btnNew);
          toolBar.add(btnSave);
          toolBar.add(btnOpen);
          toolBar.add(btnSQLExecute);
          toolBar.add(btnDFQLExecute);

//*******************************************************************
         //End ToolBar
//*******************************************************************

//*******************************************************************
         //Next Buttons
//*******************************************************************

         btnNext1.setToolTipText("dispalys next operator\'s results");
         btnNext2.setToolTipText("dispalys next database\'s results");

         btnNext1.setBounds(new Rectangle((int)(WIDTH/10)+
                           10,2*(int)(HEIGHT*5/13)+17,
                           (int)(WIDTH*5/16),20));

         btnNext2.setBounds(new Rectangle((int)(WIDTH/10)+
                           10,2*(int)(HEIGHT*5/13)+39,
                           (int)(WIDTH*5/16),20));

         btnNext1.setEnabled(false);
         btnNext1.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(ActionEvent e) {
           btnNext1_actionPerformed(e);
          }});

         btnNext2.setEnabled(false);
         btnNext2.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(ActionEvent e) {
           btnNext2_actionPerformed(e);
          }});

//*******************************************************************
         //End Next Buttons
//*******************************************************************
```

```
//*******************************************************************
        //Menu
//*******************************************************************

        //Menu File
        menuFile            = new JMenu     ("File");
        menuFileExit        = new JMenuItem ("Exit");
        menuFilePrint       = new JMenuItem ("Print");

        menuFileExit.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(ActionEvent e) {
          menuFileExit_actionPerformed(e);
         }});

        menuFilePrint.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(ActionEvent e) {
          menuFilePrint_actionPerformed(e);
         }});

        menuFile.add(menuFileExit);
        menuFile.add(menuFilePrint);

        //Menu Connection
        menuConnection                 = new JMenu     ("Connection");
        menuConnectionConnect          = new JMenuItem ("Connect");
        menuConnectionDisConnect       = new JMenuItem ("Disconnect");

        menuConnectionConnect.addActionListener(new
              java.awt.event.ActionListener() {
         public void actionPerformed(ActionEvent e) {
          menuConnectionConnect_actionPerformed(e);
         }});

        menuConnectionDisConnect.addActionListener(new
              java.awt.event.ActionListener() {
         public void actionPerformed(ActionEvent e) {
          menuConnectionDisConnect_actionPerformed(e);
         }});

        menuConnection.add(menuConnectionConnect);
        menuConnection.add(menuConnectionDisConnect);
        menuConnection.add(menuSettings);

        //Menu SQL
        menuSQL             = new JMenu     ("SQL");
```

```
menuSQLExecute  = new JMenuItem ("Execute");

menuSQLExecute.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    menuSQLExecute_actionPerformed(e);
  }});
menuSQL.add(menuSQLExecute);

//Menu DFQL
menuDFQL             = new JMenu     ("DFQL");
menuDFQLNew          = new JMenuItem ("New");
menuDFQLSave         = new JMenuItem ("Save");
menuDFQLOpen         = new JMenuItem ("Load");
menuDFQLExecute      = new JMenuItem ("Execute");

menuDFQLNew.addActionListener(new java.awt.event.ActionListener(){
  public void actionPerformed(ActionEvent e) {
    menuDFQLNew_actionPerformed(e);
  }});

menuDFQLSave.addActionListener(new java.awt.event.ActionListener(){
  public void actionPerformed(ActionEvent e) {
    menuDFQLSave_actionPerformed(e);
  }});

menuDFQLOpen.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    menuDFQLOpen_actionPerformed(e);
  }});

menuDFQLExecute.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    menuDFQLExecute_actionPerformed(e);
  }});

menuDFQL.add(menuDFQLNew);
menuDFQL.add(menuDFQLSave);
menuDFQL.add(menuDFQLOpen);
menuDFQL.addSeparator();
menuDFQL.add(menuDFQLExecute);

//Menu Help
menuHelp          = new JMenu     ("Help");
menuHelpManual    = new JMenuItem ("Manual");
```

```
      menuHelpManual.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          menuHelpManual_actionPerformed(e);
        }});
      menuHelp.add(menuHelpManual);

      //Adding menubar
      menuBar = new JMenuBar();
      menuBar.add(menuFile);
      menuBar.add(menuConnection);
      menuBar.add(menuSQL);
      menuBar.add(menuDFQL);
      menuBar.add(menuHelp);

//*************************************************************
      //End menu
//*************************************************************

//*************************************************************
      //Operator Tool Bar
//*************************************************************

      //toolBasic

      toolBasic.setBounds((int)(WIDTH*33/80)+20,(int)(HEIGHT*11/15)+16,
                (int)(WIDTH/80*43),(int)(HEIGHT/10)-5);
      btnSelect.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          btnSelect_actionPerformed(e);
        }});

      btnProject.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          btnProject_actionPerformed(e);
        }});

      btnJoin.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          btnJoin_actionPerformed(e);
        }});

      btnUnion.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
          btnUnion_actionPerformed(e);
        }});
```

```
btnDiff.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    btnDiff_actionPerformed(e);
  }});

btnGroupCnt.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    btnGroupCnt_actionPerformed(e);
  }});

toolBasic.add(btnSelect);
toolBasic.add(btnProject);
toolBasic.add(btnJoin);
toolBasic.add(btnUnion);
toolBasic.add(btnDiff);
toolBasic.add(btnGroupCnt);

//toolAdvance
toolAdvance.setBounds((int)(WIDTH*33/80)+20,(int)(HEIGHT*11/15)+16,
                      (int)(WIDTH/80*43),(int)(HEIGHT/10)-5);


btnEqjoin.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(ActionEvent e) {
   btnEqjoin_actionPerformed(e);
  }});

btnGrpAllSat.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(ActionEvent e) {
   btnGrpAllSat_actionPerformed(e);
  }});

btnGrpNSat.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(ActionEvent e) {
   btnGrpNSat_actionPerformed(e);
  }});

btnGrpMax.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(ActionEvent e) {
   btnGrpMax_actionPerformed(e);
  }});

btnGrpMin.addActionListener(new java.awt.event.ActionListener() {
public void actionPerformed(ActionEvent e) {
   btnGrpMin_actionPerformed(e);
  }});
```

```
        btnGrpAvg.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(ActionEvent e) {
           btnGrpAvg_actionPerformed(e);
          }});

        btnIntersect.addActionListener(new java.awt.event.ActionListener() {
        public void actionPerformed(ActionEvent e) {
           btnIntersect_actionPerformed(e);
          }});

        toolAdvance.add(btnEqjoin);
        toolAdvance.add(btnGrpAllSat);
        toolAdvance.add(btnGrpNSat);
        toolAdvance.add(btnGrpMax);
        toolAdvance.add(btnGrpMin);
        toolAdvance.add(btnGrpAvg);
        toolAdvance.add(btnIntersect);

        //toolUser1
        toolUser1.setBounds((int)(WIDTH*33/80)+20,(int)(HEIGHT*11/15)+16,
                                (int)(WIDTH/80*43),(int)(HEIGHT/10)-5);

        userNew.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(ActionEvent e) {
           userNew_actionPerformed(e);
          }});

        userLoad.addActionListener(new java.awt.event.ActionListener() {
          public void actionPerformed(ActionEvent e) {
           userLoad_actionPerformed(e);
          }});

        toolUser1.add(userNew);
        toolUser1.add(userLoad);

//*****************************************************************
        //End Operators tool bar
//*****************************************************************

//*****************************************************************
        //General configuration
//*****************************************************************

        // color
        cnvDFQL.setBackground   (SystemColor.info);
```

```
txtSQL.setBackground      (Color.black);
txtSQL.setForeground      (Color.white);
metaData.setBackground    (Color.pink);

// enable / editable
metaData.setEditable      (false);
panelOperators.setVisible (false);
panelMain.setEnabled      (false);

this.setJMenuBar            (menuBar);
metaData.setFont            (new java.awt.Font("Monospaced", 1, 12));
SQLResult.setAutoResizeMode       (JTable.AUTO_RESIZE_OFF);
treeMeta.addTreeSelectionListener (new DBSelectionListener());
treeMeta.setModel            (null);

panelMain.add (txtSQL, "SQL");
panelMain.add (cnvDFQL, "DFQL");
panelMain.add (metaData, "Definition");
panelMain.add (panelTableData, "Data");

panelOperators.add  (toolBasic,"Basic");
panelOperators.add  (toolAdvance,"Advance");
panelOperators.add  (toolUser1,"User-Defined");

panelMain.setBounds(new Rectangle((int)(WIDTH*33/80)+20, 10,
                                  (int)(WIDTH/80*43),
                                  (int)(HEIGHT*11/15)+6));

panelOperators.setBounds(new Rectangle ((int)(WIDTH*33/80)+20,
                                        (int)(HEIGHT*11/15)+16,
                                        (int)(WIDTH/80*43),
                                        (int)(HEIGHT/10)));

panelMeta.setBounds(new Rectangle((int)(WIDTH/10)+10, 15,
                                  (int)(WIDTH*5/16),
                                  (int)(HEIGHT*5/13)));

panelResults.setBounds(new Rectangle((int)(WIDTH/10)+10,
                                     (int)(HEIGHT*5/13)+16,
                                     (int)(WIDTH*5/16),
                                     (int)(HEIGHT*5/13)));

this.getContentPane().add(toolBar, null);
this.getContentPane().add(panelMeta);
this.getContentPane().add(panelResults, null);
this.getContentPane().add(panelMain, null);
```

```
  this.getContentPane().add(panelOperators,null);
  this.getContentPane().add(btnNext1,null);
  this.getContentPane().add(btnNext2,null);
  this.getContentPane().setLayout(null);
  panelMain.setSelectedIndex(0);
  generateTree();
  design = false;
 }

/*              WINDOW ACTIVATIONS                    */
/**
  * The window listener methods
  * @param event (WindowEvent)
**/
public void windowClosing          (WindowEvent event){System.exit(0);}
public void windowActivated        (WindowEvent event){}
public void windowClosed           (WindowEvent event){}
public void windowDeactivated      (WindowEvent event){}
public void windowDeiconified      (WindowEvent event){}
public void windowIconified        (WindowEvent event){}
public void windowOpened           (WindowEvent event){}

/*              MENU ACTION PERFORMED                     */
/**
  * Exit menu action performed
  * @param e (ActionEvent)
**/
void menuFileExit_actionPerformed(ActionEvent e) {
 System.exit(1);
}

/**
  * Print menu action performed
  * @param e (ActionEvent)
**/
void menuFilePrint_actionPerformed(ActionEvent e) {
 print();
}

/**
  * Manual menu action performed
  * @param e (ActionEvent)
**/
void menuHelpManual_actionPerformed(ActionEvent e) {
 help = new Help();
}
```

```
/**
  * Connect menu action performed
  * @param e (ActionEvent)
**/
void connect(){
  Logon logon  = new Logon();
}

/**
  * Disconnect menu action performed
  * @param e (ActionEvent)
**/
void disconnect(){
 if (databaseList.size() != 0){
   try {
     databaseList.removeElement(currentDatabase);
     String oldURL = currentDatabase.getURL();
     currentDatabase.disconnect();

     if (databaseList.size() != 0){
       currentDatabase = (Database) databaseList.lastElement();
       this.setTitle(currentDatabase.getURL());
     }
     else {
       currentDatabase = null;
       this.setTitle("No Current Database Connection" );
       panelMain.setEnabled(false);
       panelOperators.setVisible(false);
       metaData.setText(" ");
       relation.clear();
       clear();
     }
     generateTree();
     message.setMessage("disconnected : " + oldURL);
     message.show(true);
   }
   catch (SQLException e) {
     message.setMessage(e.getMessage()) ;
     message.show(true);
   }
 }
}
```

```
/**
  * Connect menu action performed
  * @param e (ActionEvent)
**/
void menuConnectionConnect_actionPerformed(ActionEvent e) {
 connect();
}

/**
  * Disconnect menu action performed
  * @param e (ActionEvent)
**/
void menuConnectionDisConnect_actionPerformed(ActionEvent e) {
 disconnect();
}

/**
  * Settings menu action performed
  * @param e (ActionEvent)
**/
void menuSettings_actionPerformed(ActionEvent e) {
 if (currentDatabase != null){
   Settings settings = new Settings(currentDatabase);
   settings.show(true);
 }
}

/**
  * SQLExecute menu action performed
  * @param e (ActionEvent)
**/
void menuSQLExecute_actionPerformed(ActionEvent e) {
 SQLExecute();
}

/**
  * New menu action performed
  * @param e (ActionEvent)
**/
void menuDFQLNew_actionPerformed(ActionEvent e) {
 clear();
}
```

```
 /**
  * Save menu action performed
  * @param e (ActionEvent)
 **/
 void menuDFQLSave_actionPerformed(ActionEvent e) {
    save();
 }

 /**
  * Open menu action performed
  * @param e (ActionEvent)
 **/
 void menuDFQLOpen_actionPerformed(ActionEvent e) {
   open();
 }

 /**
  * DFQLExecute menu action performed
  * @param e (ActionEvent)
 **/
 void menuDFQLExecute_actionPerformed(ActionEvent e) {
  index = 0;
  DFQLExecute();
 }

 /*              TOOL BUTTONS ACTION PERFORMED                    */
 /**
  * Exit button action performed
  * @param e (ActionEvent)
 **/
 void btnExit_actionPerformed(ActionEvent e) {
  System.exit(1);
 }

 /**
  * Print button action performed
  * @param e (ActionEvent)
 **/
 void btnPrint_actionPerformed(ActionEvent e) {
  print();
 }

 /**
  * Connect button action performed
  * @param e (ActionEvent)
 **/
```

```
void btnConnect_actionPerformed(ActionEvent e) {
  connect();
}

/**
  * Disconnect button action performed
  * @param e (ActionEvent)
**/
void btnDisConnect_actionPerformed(ActionEvent e) {
  disconnect();
}

/**
  * SQLExecute button action performed
  * @param e (ActionEvent)
**/
void btnSQLExecute_actionPerformed(ActionEvent e) {
  SQLExecute();
}

/**
  * New button action performed
  * @param e (ActionEvent)
**/
void btnNew_actionPerformed(ActionEvent e) {
  clear();
}

/**
  * Save button action performed
  * @param e (ActionEvent)
**/
void btnSave_actionPerformed(ActionEvent e) {
  save();
}

/**
  * Open button action performed
  * @param e (ActionEvent)
**/
void btnOpen_actionPerformed(ActionEvent e) {
  open();
}
```

```
/**
  * DfqlExecute button action performed
  * @param e (ActionEvent)
**/
void btnDFQLExecute_actionPerformed(ActionEvent e) {
  index = 0;
  DFQLExecute();
}

/**
  * Manual button action performed
  * @param e (ActionEvent)
**/
void btnManual_actionPerformed(ActionEvent e) {
  help = new Help ();
}

/**
  * Send all button action performed
  * @param e (ActionEvent)
**/
void btnSendAll_actionPerformed(ActionEvent e) {
  index=0;
  indexDatabase = 0;
  sendToAllDatabases();
}

/*          OPERATOR BUTTONS ACTION PERFORMED               */
/**
  * Select button action performed
  * @param e (ActionEvent)
**/
void btnSelect_actionPerformed(ActionEvent e) {
  OperatorSelect select = new OperatorSelect(20*Counter[0],70);
}

/**
  * Project button action performed
  * @param e (ActionEvent)
**/
void btnProject_actionPerformed(ActionEvent e) {
  OperatorProject project = new OperatorProject(20*Counter[1],70);
}
```

```
/**
 * Join button action performed
 * @param e (ActionEvent)
**/
void btnJoin_actionPerformed(ActionEvent e) {
  OperatorJoin join = new OperatorJoin(20*Counter[2],70);
}

/**
 * Union button action performed
 * @param e (ActionEvent)
**/
void btnUnion_actionPerformed(ActionEvent e) {
  OperatorUnion union = new OperatorUnion(20*Counter[3],70);
}

/**
 * Different button action performed
 * @param e (ActionEvent)
**/
void btnDiff_actionPerformed(ActionEvent e) {
  OperatorDiff diff = new OperatorDiff(20*Counter[4],70);
}

/**
 * Group count button action performed
 * @param e (ActionEvent)
**/
void btnGroupCnt_actionPerformed(ActionEvent e) {
  OperatorGroupCnt gCount = new OperatorGroupCnt(20*Counter[5],70);
}

/**
 * Eqjoin button action performed
 * @param e (ActionEvent)
**/
void btnEqjoin_actionPerformed(ActionEvent e) {
  OperatorEqjoin select = new OperatorEqjoin(20*Counter[6],70);
}

/**
 * Group all satisfy button action performed
 * @param e (ActionEvent)
**/
void btnGrpAllSat_actionPerformed(ActionEvent e) {
  OperatorGrpAllSat grpAllSat = new OperatorGrpAllSat(20*Counter[7],70);
}

/**
 * Group number satisfy button action performed
 * @param e (ActionEvent)
**/
void btnGrpNSat_actionPerformed(ActionEvent e) {
  OperatorGrpNSat grpNSat = new OperatorGrpNSat(20*Counter[8],70);
}

/**
 * Group maximum button action performed
 * @param e (ActionEvent)
**/
void btnGrpMax_actionPerformed(ActionEvent e) {
  OperatorGrpMax grpMax = new OperatorGrpMax(20*Counter[9],70);
}

/**
 * Group minimum button action performed
 * @param e (ActionEvent)
**/
void btnGrpMin_actionPerformed(ActionEvent e) {
  OperatorGrpMin grpMin = new OperatorGrpMin(20*Counter[10],70);
}

/**
 * Group average button action performed
 * @param e (ActionEvent)
**/
void btnGrpAvg_actionPerformed(ActionEvent e) {
  OperatorGrpAvg grpAvg = new OperatorGrpAvg(20*Counter[11],70);
}

/**
 * Intersect button action performed
 * @param e (ActionEvent)
**/
void btnIntersect_actionPerformed(ActionEvent e) {
  OperatorIntersect intersect = new OperatorIntersect(20*Counter[12],70);
}

/**
 * New user defined Operator button action performed
 * @param e (ActionEvent)
**/
```

```
void userNew_actionPerformed(ActionEvent e) {
  designRelation      = new Vector();
  designCondition     = new Vector();
  designAttribute     = new Vector();
  designOps           = new Vector();
  designQueries       = new Vector();

  UserOperator userOperator = new UserOperator();
  setDesign (true);
}

/**
 * Load user defined Operator button action performed
 * @param e (ActionEvent)
**/
void userLoad_actionPerformed(ActionEvent e) {
  OpenFile loadUSO = new OpenFile(true);
}

/**
 * SQLExecute, sends the SQL query to the database currently active,
 * to be executed.
**/
void SQLExecute(){
 if (currentDatabase != null)
 {
  sqlQuery = txtSQL.getText();
  try
  {
   Table deneme = currentDatabase.executeQuery(sqlQuery);
   relation.assignData(deneme.getData(), deneme.getAttributes());
  }
  catch (SQLException ex)
  {
   message.setMessage(ex.getMessage());
   message.show(true);
   txtSQL.grabFocus();
  }
 }
}

/**
 * DFQLExecute, sends the DFQL query to the database currently active,
 * to be executed. DFQL query consists of one or more operators. Each operator
 * produces a query of its own. According to the user choice, the result of an
 * operator can be displayed. This method, displays all the operators query
```

```
  * results, which are selected to be displayed, one by one.
 **/
 void DFQLExecute(){
  if (currentDatabase != null){
     for(int xi = index ; xi < DFQLQuery.size();xi++){
      tempOperator = (Operator)DFQLQuery.elementAt(xi);
      if (tempOperator.display){
        btnNext1.setText("Displaying " + tempOperator.name);
        sqlQuery = tempOperator.query;
        if(xi!=DFQLQuery.size()-1){
          btnNext1.setEnabled(true);
        }
        index = xi;
        DFQLDisplay();
        break;
      }
     }
  }
 }

/**
  * DFQLDisplay, displays the results of the operator queries.
**/
public void DFQLDisplay(){
   try{
     Table deneme = currentDatabase.executeQuery(sqlQuery);

     txtSQL.setText(sqlQuery);
     relation.assignData(deneme.getData(),
                  deneme.getAttributes());
   }
   catch (SQLException ex){
     message.setMessage(ex.getMessage());
     message.show(true);
     txtSQL.grabFocus();
   }
}

/**
  * Next 1 button is used to switch from one operators result to the following
**/
void btnNext1_actionPerformed(ActionEvent e){
  index++;
  btnNext1.setEnabled(false);
  DFQLExecute();
}
```

```
/**
  * Next 2 button is used to switch from one database to the following one
**/
void btnNext2_actionPerformed(ActionEvent e){
  index = 0;
  indexDatabase++;
  btnNext2.setEnabled(false);
  sendToAllDatabases();
}

/**
  * Clear, removes all elements of the vectors, initiates all the variables,
  * clears all the panel, tables and etc.
**/
public void clear() {
 cnvDFQL.clear();
 for (int xi = 0; xi<13;xi++){
   Counter[xi] = 1;
 }
 DFQLQuery.removeAllElements();
 txtSQL.setText("");
 btnNext1.setText("DISPLAY OPERATOR");
 btnNext2.setText("DISPLAY DATABASE");
}

/**
  * Creates a print frame to print the result
**/
public void print(){
   PrintPreview print = new PrintPreview(SQLResult);
   print.print( );
}

/**
  * Creates a open file frame to load a previously saved dfql query
**/
public void open(){
 if (currentDatabase != null){
   OpenFile open = new OpenFile();
 }
}

/**
  * Creates a save file frame to save the current dfql query
**/
```

```
public void save(){
  saveConfig save = new saveConfig();
 }

 /**
  * This method is used to generate a tree which shows database meta data
  * at every connection or disconnection this tree is updated.
 **/
void generateTree () throws SQLException{
   DefaultTreeModel result   = null;
   int  databaseCount           = databaseList.size();
   DefaultMutableTreeNode    top = new DefaultMutableTreeNode
                                               ("Database URLs");
   DefaultMutableTreeNode parent = top;
   Database              tempDB;

   for (int index = 0 ; index < databaseCount; index++){
     tempDB     = (Database) databaseList.elementAt(index);
     parent.add(tempDB.getTreeModel());
   }

   result = new DefaultTreeModel(top);
   treeMeta.setModel(result);
}

/**
  * This method is used to send the prepared DFQL or SQL query to the
  * currently
  * connected databases. It starts from the first database connected and
  * switchs to the next one every time next2 button is pressed.
**/
void sendToAllDatabases(){
  for (int ix = indexDatabase; ix<databaseList.size();ix++){
   currentDatabase = (Database)databaseList.elementAt(ix);
   this.setTitle(currentDatabase.getURL());
   btnNext2.setText("Displaying " + currentDatabase.getURL());
   if(ix<databaseList.size()-1){
     btnNext2.setEnabled(true);
   }

   if (indexPanel == 0){
     SQLExecute();
     break;
   }

   if (indexPanel == 1){
    index = 0;
    DFQLExecute();
    break;
    }
  }
}

// DBSelectionListener inner class
/**
  * This inner class is used to add a listener to the database tree. This listener
  * allows user to select a database or table of a database or attribute of a table.
  * according to the selection, the information presented about the database
  * changes.
**/
class DBSelectionListener implements TreeSelectionListener{

 public void valueChanged(TreeSelectionEvent e){
   TreePath path    = e.getNewLeadSelectionPath();
   TreePath oldPath = e.getOldLeadSelectionPath();
   if (path != null){
     DefaultMutableTreeNode tempNode;
     Object [] nodes            = path.getPath();
     DefaultMutableTreeNode node = (DefaultMutableTreeNode)
                                         nodes[nodes.length-1];
     String selectedName        = (String) node.getUserObject();
     panelMain.setSelectedIndex(2);  // Activate the Definition pane

     if (currentDatabase != null)  {
      if (nodes.length > 1) {
        DefaultMutableTreeNode nodeDB = (DefaultMutableTreeNode)
                                               nodes[1];
        String selectedDB = (String) nodeDB.getUserObject();
        for (int index = 0; index<databaseList.size();index++){
         Database temp = (Database) databaseList.elementAt(index);
         if (selectedDB.equalsIgnoreCase(temp.getURL())) {
          if (!(selectedDB.equalsIgnoreCase(currentDatabase.getURL()))) {
            currentDatabase = temp;
            treeMeta.collapsePath(oldPath);
            break;
          }
         }
        }
      }

      try {
       switch (nodes.length) {
```

```
 case 2 :
  metaData.setText(" URL          :  " + currentDatabase.getURL()+ "\n");
  metaData.append (" DRIVER : " + currentDatabase.getDriver () + "\n");
  metaData.append (" USERID"+currentDatabase.getUserName()+ "\n");
  metaData.append (" PROGRAM NAME   :  " +
                        currentDatabase.getDatabaseProgramName()+"\n");
  metaData.append (" FULL ANSI92    :  " +
                        currentDatabase.isSupportANSI92()+ "\n");
  test.window.setTitle(currentDatabase.getURL());
  break;
 case 3 :
  tempNode = (DefaultMutableTreeNode) nodes[2];
  Vector atts = currentDatabase.getTableAttributes
                ((String)tempNode.getUserObject());
  Vector tableData = currentDatabase.getTableData
                        ((String)tempNode.getUserObject());
  panelMain.setSelectedIndex(3);
  relationTable.assignData(tableData, atts);
  break;
 case 4 :
  Vector  attrs;
  Attribute tempAttr;
  tempNode = (DefaultMutableTreeNode) nodes[2];
  attrs = currentDatabase.getTableAttributes
                        ((String)tempNode.getUserObject());
  for (int attIndex = 0; attIndex < attrs.size(); attIndex++) {
     tempAttr = (Attribute)attrs.elementAt(attIndex);
     if (selectedName.equalsIgnoreCase(tempAttr.getAttributeName())) {
        metaData.setText(" TYPE  :" + tempAttr.getAttributeType()+"\n");
        metaData.append (" SIZE :  " + tempAttr.getAttributeSize()+ "\n");
        metaData.append (" NULLABLE :" + tempAttr.isNullable()+"\n");
        metaData.append (" SEARCHABLE : " +
                           tempAttr.isSearchable()+"\n");
        metaData.append (" CASE SENS. :  " +
                           tempAttr.isCaseSensitive()+"\n");
     break;
     }
  }
  break;

 default : break;
 }
}
catch (SQLException exs) {
   message.setMessage(exs.getMessage());
   message.show(true);
```

```
          }
        }
      }
    }
  }

  void txtSQL_componentShown(ComponentEvent e) {
    arrangeDisplay();
  }

  void metaData_componentShown(ComponentEvent e) {
    arrangeDisplay();
  }

  void panelTableData_componentShown(ComponentEvent e) {
    arrangeDisplay();
  }

  void denememe_componentShown(ComponentEvent e) {
    arrangeDisplay();
  }

  /**
   * In order to switch between the panels. If DFQL panel is selected, then
   * Operators Panel becomes visible.
  **/
  void arrangeDisplay () {
    int selIndex = panelMain.getSelectedIndex();
    if (selIndex == 1){
      panelOperators.setVisible(true);
    }
    else {
      panelOperators.setVisible(false);
    }
    indexPanel = selIndex;
  }

  /**
   Changes the mode of the program
  **/
  void setDesign (boolean newDesign){
    design = newDesign;
  }
```

```
 /**
   Returns the mode of the program
  **/
  boolean isDesign (){
   return design;
  }

  String getProgramPath(){
   return path;
  }
}
```

10. Message.Java

```
package ThesisGUI;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

/**
 * This frame is used to display error, warning, exception and information
 * messages, such as "no connection established"
 * AUTHOR : IS & BA
**/
public class Message extends JFrame {
 /** Scroll panel for message text area **/
 JScrollPane panelError  = new JScrollPane();

 /** Text area to display the messages **/
 JTextArea txtError      = new JTextArea();

 /** close button to close this frame**/
 JButton btnClose        = new JButton();

 /*              CONSTRUCTOR                    */
 /**
  * The constructor
 **/
 public Message() {
  try {
   jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
 }
/*              INITIATE METHOD                    */
/**
  * The initiation
  * creates the frame, close button, text area and scroll pane. Initiates them.
**/

private void jbInit() throws Exception {

  this.setTitle   ("Message");
  btnClose.setText("Close");

  panelError.setBounds(new Rectangle(10, 10, 300, 50));
  btnClose.setBounds  (new Rectangle(120, 65, 70, 25));
  txtError.setEditable(false);

  btnClose.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
      btnClose_actionPerformed(e);
    }});

  this.getContentPane().setLayout    (null);
  this.getContentPane().add          (panelError, null);
  this.getContentPane().add          (btnClose, null);
  panelError.getViewport().add   (txtError, null);
  setBounds(350,200,330,120);
}

/** Changes the contents of the text area with new message
  * @param message (String)
**/
void setMessage(String message){
   txtError.setText(message);  }

/** Terminates this frame
  * @param message e (ActionEvent)
**/
  void btnClose_actionPerformed(ActionEvent e){
  this.dispose();
 }
}
```

11. OpenFile.Java

```
package ThesisGUI;
```

```
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.io.*;

/**
 * This frame is designed to be used to open previously designed queries.
 * In other words, It is basically for the user defined operators.
 * AUTHOR : IS & BA
**/
public class OpenFile extends JFrame {
 /** File Dialog object for loading a file **/
 FileDialog file = new FileDialog(this,"Load File",FileDialog.LOAD);

 /**
  * Boolean TRUE  : if user wants to load a user defined operator
  *        FALSE : if user wants to load a pre-executed and saved DFQL query
  **/
 boolean isUSO = false;

 /*            CONSTRUCTORS                     */
 /**
  * The default constructor
 **/
 public OpenFile() {
  try {
   jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
 }

 /**
  * The constructor to create an instance for loading a user defined operator
  * @param yes boolean
 **/
 public OpenFile(boolean yes) {
  try {
   isUSO = yes;
   jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
 }
```

```
/*              INITIATE METHOD                 */
/**
  * The initiation
  * creates the file dialog object. And inititates it.
  * Later on calls load config method.
 **/
 private void jbInit() throws Exception {
  this.setResizable (false);

  if (isUSO){
   file.setDirectory (test.window.getProgramPath()+"\\operator\\");
   file.setFile     ("*.uso");
   loadUSO();
  }
  else{
   file.setDirectory (test.window.getProgramPath()+"\\dfql\\");
   file.setFile     ("*.dfql");
   loadConfig();
  } }

/**
  * This method is used to configure the OpenFile object to load a DFQL query.
  * Checks the name of the file. If a file is selected. It calls the transfer
  * method. otherwise just cancel the job.
**/
void loadConfig(){
 file.show(true);
 String fileName = file.getFile();
 if (fileName != null){
  fileName = file.getDirectory() + fileName;
  transfer(fileName);
 }
}

/**
 * Creates the ".dfql" file objects , which are going to be used during file
 * loading. the first line of the file specifies the length of the file.
 * According to this number it reads every line, creates a new operator
 * object with this information and calls add method of the draw class.
 * @param fileName (String)
**/
void transfer (String fileName){
 try{
   File newFile           = new File (fileName);
   FileReader inStream    = new FileReader (newFile);
```

```
BufferedReader bufFile  = new BufferedReader (inStream);
String line             = bufFile.readLine();
int number              = Integer.parseInt(line);
Operator operator1      = new Operator();

for(int xi = 0; xi<number;xi++){
 line = bufFile.readLine();
 if (line.equalsIgnoreCase("select") ){
  test.window.Counter[0] = test.window.Counter[0] + 1;
 }
 else if (line.equalsIgnoreCase("project") ){
  test.window.Counter[1] = test.window.Counter[1] + 1;
 }
 else if (line.equalsIgnoreCase("diff") ){
  test.window.Counter[4] = test.window.Counter[4] + 1;
 }
 else if (line.equalsIgnoreCase("eqjoin") ){
  test.window.Counter[6] = test.window.Counter[6] + 1;
 }
 else if (line.equalsIgnoreCase("groupcnt") ){
  test.window.Counter[5] = test.window.Counter[5] + 1;
 }
 else if (line.equalsIgnoreCase("grpallsat") ){
  test.window.Counter[7] = test.window.Counter[7] + 1;
 }
 else if (line.equalsIgnoreCase("grpavg") ){
  test.window.Counter[11] = test.window.Counter[11] + 1;
 }
 else if (line.equalsIgnoreCase("grpmax") ){
  test.window.Counter[9] = test.window.Counter[9] + 1;
 }
 else if (line.equalsIgnoreCase("grpmin") ){
  test.window.Counter[10] = test.window.Counter[10] + 1;
 }
 else if (line.equalsIgnoreCase("grpnsat") ){
  test.window.Counter[8] = test.window.Counter[8] + 1;
 }
 else if (line.equalsIgnoreCase("intersect") ){
  test.window.Counter[12] = test.window.Counter[12] + 1;
 }
 else if (line.equalsIgnoreCase("join") ){
  test.window.Counter[2] = test.window.Counter[2] + 1;
 }
 else if (line.equalsIgnoreCase("union") ){
  test.window.Counter[3] = test.window.Counter[3] + 1;
 }
```

```
          operator1.setType(line);
          line = bufFile.readLine();

          operator1.setName(line);
          line = bufFile.readLine();

          operator1.setX(Integer.parseInt(line));
          line = bufFile.readLine();

          operator1.setY(Integer.parseInt(line));
          line = bufFile.readLine();

          operator1.setNumberOfInputNodes(Integer.parseInt(line));
          line = bufFile.readLine();

          if (line.equals("true")){
            operator1.setDisplay(true);
          }

          else{
            operator1.setDisplay(false);
          }

          line = bufFile.readLine();
          operator1.setQuery(line);

          String [] input = {"none","none","none","none"};

          for(int di=0;di<(operator1.numberOfInputNodes);di++){
            line = bufFile.readLine();
            input[di] = line;
          }

          test.window.cnvDFQL.add(operator1.x,operator1.y,
                        operator1.name, operator1.numberOfInputNodes,
                        input, operator1.display, operator1.type,
                        operator1.query);
        }
        bufFile.close();
      }
      catch (IOException e){
        test.window.message.setMessage(e.getMessage());
        test.window.message.show(true);
      }
    }
```

```
/**
  * This method is used to configure the OpenFile object to load a user
  * defined operator. Checks the name of the file. If a file is selected,
  * it reads the file and creates an instance of the OperatorUser class
  * using the data stored in the file
**/
void loadUSO(){
  int x, y, rel, con, att;
  String query, type;
  file.show(true);
  String fileName = file.getFile();
  if (fileName != null){
   fileName = file.getDirectory() + fileName;

 try{
     File newFile              = new File (fileName);
     FileReader inStream       = new FileReader (newFile);
     BufferedReader bufFile = new BufferedReader (inStream);
     String line               = bufFile.readLine();
     x                         = Integer.parseInt(line);
     line                      = bufFile.readLine();
     y                         = Integer.parseInt(line);
     line                      = bufFile.readLine();
     rel                       = Integer.parseInt(line);
     line                      = bufFile.readLine();
     con                       = Integer.parseInt(line);
     line                      = bufFile.readLine();
     att                       = Integer.parseInt(line);
     line                      = bufFile.readLine();
     query                     = line;
     line                      = bufFile.readLine();
     type                      = line;

     OperatorUser loaded = new OperatorUser(x,y,rel,con,att,query,type);

     bufFile.close();
 }
 catch (IOException e){
   test.window.message.setMessage(e.getMessage());
   test.window.message.show(true);
 }}
}
}
```

12. Operator.Java

```
package ThesisGUI;

/**
 * This is one of the main class of our design. It holds all the information,
 * we need to know about the operator.
 * AUTHOR : IS & BA
**/

public class Operator {
 /** X coordinate of the operator **/
 protected int x;

 /** Y coordinate of the operator **/
 protected int y;

 /** The number of the input nodes of the operator **/
 protected int numberOfInputNodes;


 /** name of the operator **/
 protected String name;

 /** Contents of the input nodes of the operator **/
 protected String [] input = new String[4];

 /** Display info for the operator **/
 protected boolean display;

 /** Type of the operator **/
 protected String type;

 /** SQL query that this operator represents **/
 protected String query;

 /*          CONSTRUCTOR                */
 /**
  * Default constructor
 **/
 public Operator(){ }

 /*          CONSTRUCTOR                */
 /**
  * Constructor with initial parameters
  * @ parameters x, y, and number (integer),
 * name, []input, type and query (String),
 * display                 (boolean)
**/
public Operator(int x, int y, String name, int number, String [] input,
            boolean display, String type, String query) {
 this.x                    = x;
 this.y                    = y;
 this.name                 = name;
 this.numberOfInputNodes   = number;
 this.display              = display;
 this.type                 = type;
 this.query                = query;
 this.input[0]             = input[0];
 this.input[1]             = input[1];
 this.input[2]             = input[2];
 this.input[3]             = input[3];
}

/*          Set Methods                 */
/**
 * Sets the value of the X coordinate
 * @parameter X (integer)
**/
public void setX(int x){
 this.x = x;
}

/**
 * Sets the value of the Y coordinate
 * @parameter Y (integer)
**/
public void setY(int y){
 this.y = y;
}

/**
 * Sets the operator name
 * @parameter name (String)
**/
public void setName(String name){
 this.name = name;
}

/**
 * Sets the number of the input nodes of the operator
 * @parameter number (integer)
```

```
**/
public void setNumberOfInputNodes(int number){
 this.numberOfInputNodes = number;
}

/**
  * Sets the value of the display
  * @parameter display (boolean)
**/
public void setDisplay(boolean display){
 this.display = display;
}

/**
  * Sets the type of the operator
  * @parameter type (String)
**/
public void setType(String type){
 this.type = type;
}

/**
  * Sets the input values of the operator
  * @parameter input ([]String)
**/
public void setInput (String [] input){
 for(int xi =0; xi<numberOfInputNodes;xi++){
   this.input[xi] = input[xi];
 }
}

/**
  * Sets the sql query of the operator
  * @parameter query (String)
**/
public void setQuery (String query){
 this.query = query;
}

/**
  * Gets the value of the input nodes of the operator
  * @parameter input ([]String)
**/
public String [] getInput (){
 return input;
}}
```

13. OperatorDiff.Java

```
package ThesisGUI;

import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;

/**
 * This frame is used to allow the user to enter the operator information
 * for different operator. It also allwos user to change the information
 * previously entered.
 * AUTHOR : IS & BA
**/
public class OperatorDiff extends JFrame implements WindowListener{
 /** Type of the operator (constant) **/
 final static String OPERATOR_TYPE = "Diff";

 /** Input node number of the operator (constant) **/
 final static int NO_INPUT_NODES = 3;

 /** X coordinate of the operator**/
 private int x;

 /** Y coordinate of the operator**/
 private int y;

 /** name of the operator**/
 private String name;

 /** first input node relation**/
 private String relation1;

 /** second input node relation**/
 private String relation2;

 /** name of the common attribute**/
 private String attribute;

 /** display information for the operator**/
 private boolean display = false;
```

```
/** SQL query of the operator**/
private String query;

/** Specify why this frame is called (to create a new one (false)
  * or change the current operator (true)**/
private boolean whoIsCalling = false;

/** The index of the operator in the Operators list **/
private int indexNum;

/** The index of the operator in the DFQLList**/
private int current;

/** name label of the frame**/
JLabel lblName          = new JLabel();

/** first relation label of the frame**/
JLabel lblRelation1     = new JLabel();

/** second relation label of the frame**/
JLabel lblRelation2     = new JLabel();

/** common attribute label of the frame**/
JLabel lblAttribute     = new JLabel();

/** text area to enter the name of the operator default diff#**/
JTextField txtName      = new JTextField();

/** text area to enter the name of the coomon attribute**/
JTextField txtAttribute = new JTextField();

/** Okay button to confirm the info entered**/
JButton btnOK           = new JButton();

/** check box for display information**/
JCheckBox chkResult     = new JCheckBox();

/** Combo box lists the available operators or table names that can be
  * related to this operator**/
JComboBox cmbRelation1  = new JComboBox();

/** Combo box lists the available operators or table names that can be
  * related to this operator**/
JComboBox cmbRelation2  = new JComboBox();
```

```
/** Combo box lists the available attributes that can be
  * related to this operator**/
JComboBox cmbAttribute    = new JComboBox();

/** Button cancel to cancel this operation**/
JButton btnCancel       = new JButton();

/*              CONSTRUCTOR                    */
/**
  * Default constructor
  * @parameters X and Y (integer) coordinates of the operator
**/
public OperatorDiff(int x, int y) {
  try  {
    this.x = x;
    this.y = y;
    jbInit();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}

/**
  * Constructor with all parameters
  * @parameters @ parameters x, y, caller1, caller2 (integer),
  * name,relation1, relation2                    (String),
  * display                                      (boolean)
**/
public OperatorDiff(int x, int y, int caller1, int caller2, String name,
                String relation1, String relation2, String attribute, boolean display)
{
  this.x              = x;
  this.y              = y;
  this.current        = caller1;
  this.indexNum       = caller2;
  this.whoIsCalling   = true;
  this.name           = name;
  this.relation1      = relation1;
  this.relation2      = relation2;
  this.attribute      = attribute;
  this.display        = display;

  txtName.setText(name);
  if (display){
    chkResult.setSelected(true);
```

```
 }
 else{
  chkResult.setSelected(false);
 }
 try {
  jbInit();
 }
  catch(Exception e) {
  e.printStackTrace();
 }
}

/**
 * Design constructor
 * @parameters name
**/
public OperatorDiff(String name) {
 try {
  this.name = name;
  jbInit();
 }
  catch(Exception e) {
  e.printStackTrace();
 }
}

/*              INITIATE METHOD                     */
/**
 * The initiation
 * creates the frame for user to enter the operator data, and adds
 * buttons, labels, text area and combo boxes.
**/
private void jbInit() throws Exception {
 lblName.setText            ("Name");
 this.setTitle              ("Diff");
 lblRelation1.setText       ("Relation1");
 lblRelation2.setText       ("Relation2");
 lblAttribute.setText       ("Attribute");
 chkResult.setText          ("Display");
 btnOK.setText              ("OK");
 btnCancel.setText          ("Cancel");

 lblName.setBounds          (new Rectangle(4, 2, 93, 18));
 lblRelation1.setBounds     (new Rectangle(4, 27, 93, 18));
 lblRelation2.setBounds     (new Rectangle(4, 50, 93, 18));
 lblAttribute.setBounds     (new Rectangle(4, 73, 93, 18));
```

```
txtName.setBounds          (new Rectangle(80, 2, 114, 17));
txtAttribute.setBounds     (new Rectangle(80, 73, 114, 17));
btnOK.setBounds            (new Rectangle(142, 100, 54, 22));
chkResult.setBounds        (new Rectangle(4, 100, 61, 22));
cmbRelation1.setBounds     (new Rectangle(80, 28, 114, 17));
cmbRelation2.setBounds     (new Rectangle(80, 51, 114, 17));
cmbAttribute.setBounds     (new Rectangle(80, 73, 114, 17));
btnCancel.setBounds        (new Rectangle(74, 100, 60, 23));

if (test.window.isDesign()){
 loadDesign();
}
else {
 loadSelect();
}

btnOK.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnOK_actionPerformed(e);
 }});

btnCancel.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnCancel_actionPerformed(e);
 }});

chkResult.setBorderPainted(true);
cmbRelation1.setBorder   (null);
cmbRelation2.setBorder   (null);

chkResult.setFont (new java.awt.Font("Dialog", 0, 11));
btnCancel.setFont (new java.awt.Font("Dialog", 0, 11));
btnOK.setFont     (new java.awt.Font("Dialog", 0, 11));

btnOK.setBorder     (BorderFactory.createRaisedBevelBorder());
chkResult.setBorder (BorderFactory.createRaisedBevelBorder());
btnCancel.setBorder (BorderFactory.createRaisedBevelBorder());

this.getContentPane().setLayout(null);
this.getContentPane().add(lblName, null);
this.getContentPane().add(txtName, null);
this.getContentPane().add(lblRelation1, null);
this.getContentPane().add(cmbRelation1, null);
this.getContentPane().add(cmbRelation2, null);
this.getContentPane().add(lblRelation2, null);
this.getContentPane().add(chkResult, null);
```

```
  this.getContentPane().add(btnCancel, null);
  this.getContentPane().add(btnOK, null);
  this.getContentPane().add(lblAttribute,null);
  if (test.window.isDesign()){
   this.getContentPane().add(cmbAttribute, null);
  }
  else {
   this.getContentPane().add(txtAttribute, null);
  }
  this.setBounds(100,100,210,170);
  this.show(true);
 }

/*              Set Methods                */
/**
  * Sets the first relation of the operator
  * @parameter relation1 (String)
**/
public void setRelation1 (String relation1){
 this.relation1 = relation1;
}

/**
  * Sets the second relation of the operator
  * @parameter relation2 (String)
**/
public void setRelation2 (String relation2){
 this.relation2 = relation2;
}

/**
  * Sets the common attribute of the operator
  * @parameter attribute (String)
**/
public void setAttribute (String attribute){
 this.attribute = attribute;
}

/**
  * Sets the name of the operator
  * @parameter name (String)
**/
public void setName (String name){
 this.name = name;  }

/*              Get Methods                */
```

```
/**
 * Gets the number of the input nodes of the operator
**/
public int getnumberOfNodes(){
 return this.NO_INPUT_NODES;
}

/**
 * Gets the type of the operator
**/
public String getOperatorType(){
 return this.OPERATOR_TYPE;
}

/**
 * Gets the first relationof the operator
**/
public String getRelation1 (){
 return this.relation1;
}

/**
 * Gets the second relation of the operator
**/
public String getRelation2 (){
 return this.relation2;
}

/**
 * Gets the name of the common attribute
**/
public String getAttribute (){
 return this.attribute;
}

/**
 * Gets the name of the operator
**/
public String getName (){
 return this.name;  }

/*            WINDOW ACTIVATIONS                */
/**
 * The window listener methods
 * @param event (WindowEvent)
**/
```

```
public void windowClosing       (WindowEvent event){ System.exit(0);}
public void windowActivated     (WindowEvent event){}
public void windowClosed        (WindowEvent event){}
public void windowDeactivated   (WindowEvent event){}
public void windowDeiconified   (WindowEvent event){}
public void windowIconified     (WindowEvent event){}
public void windowOpened        (WindowEvent event){}

/**
 * button okay action performed
 * Takes the information entered for the operator. Creates a new operator and
 * calls add or change method of the draw class. And sends new operator
 * information there to be added to the list.
 * @param e (ActionEvent)
**/
void btnOK_actionPerformed(ActionEvent e) {
 if(chkResult.isSelected()){
  display = true;
 }
 else{
  display = false;
 }

 setName      (txtName.getText());
 setRelation1 ((String)cmbRelation1.getSelectedItem());
 setRelation2 ((String)cmbRelation2.getSelectedItem());
 if (test.window.isDesign()){
  setAttribute((String)cmbAttribute.getSelectedItem());
 }
 else{
  setAttribute(txtAttribute.getText());
  if (txtAttribute.getText().equals("")){
   test.window.message.setMessage("You did not enter the attribute list");
   test.window.message.show(true);
  }
 }
 String [] input = {getRelation1(),getRelation2(),getAttribute(),"none","none"};

 buildQuery();

 if (test.window.isDesign()){
  test.window.designQueries.addElement(query);
 }
 else {
  if (whoIsCalling){
   test.window.cnvDFQL.change
```

```
                         (current,indexNum,x,y,getName(),getnumberOfNodes(),
                          input,display, getOperatorType(), query);
   }
   else{
    test.window.cnvDFQL.add
                         (x,y,getName(),getnumberOfNodes(),input,display,
                          getOperatorType(), query);
   }
  }
  this.dispose();
 }

/**
 * Loads the relations with the tables of the selected database and previously
 * created operators.
**/
void loadSelect() {

 if(!whoIsCalling){
  txtName.setText("Diff"+test.window.Counter[4]++);
 }

 Vector deneme = test.window.currentDatabase.getTableNames();
 for (int index = 0 ; index < deneme.size(); index++) {
   cmbRelation1.addItem((String) deneme.elementAt(index));
   cmbRelation2.addItem((String) deneme.elementAt(index));
 }

 for (int xi = 0; xi <test.window.cnvDFQL.numOfOperators;xi++){
  cmbRelation1.addItem(test.window.cnvDFQL.operators[xi].name);
  cmbRelation2.addItem(test.window.cnvDFQL.operators[xi].name);
 }
 cmbRelation1.removeItem(name);
 cmbRelation2.removeItem(name);

 if (relation1== null)
   cmbRelation1.setSelectedIndex(0);
 else
   cmbRelation1.setSelectedItem(relation1);

 if (relation2== null)
   cmbRelation2.setSelectedIndex(0);
 else
   cmbRelation2.setSelectedItem(relation2);

 txtAttribute.setText(attribute); }
```

```
/**
 * button cancel action performed
 * terminates this frame without creating an operator. decrements the counter
 * @param e (ActionEvent)
**/
void btnCancel_actionPerformed(ActionEvent e) {
   test.window.Counter[4]--;
   this.dispose();
}

/**
 * This portion of the code produces the SQL query for this operator according
 * to the information entered by user.
**/
public void buildQuery(){
  String checkRelation = isOtherOperator (relation1);
  if (checkRelation == null) {
     query = "select distinct " + attribute + " from " + relation1 +
                 " S where S." + attribute + " not in";
  }
  else {
     query = "select distinct " + attribute + " from " + "(" +
                 checkRelation + ") S where S." + attribute + " not in";
  }
  checkRelation = isOtherOperator(relation2);

  if (checkRelation == null) {
     query = query + " (select " + attribute + " from " + relation2 + ")";
  }
  else {
     query = query + "(select " + attribute + " from " + "(" +checkRelation + "))";
  }
}

/**
 * This portion of the code looks for the type of the relations, if they are
 * tables it returns null, otherwise if is another operator, it returns the
 * query of the operator which the current operator related to.
**/
public String isOtherOperator (String type) {
  String result = null;
  for (int index = 0; index < test.window.DFQLQuery.size(); index++) {
    Operator temp = (Operator) test.window.DFQLQuery.elementAt(index);
    if (type.equalsIgnoreCase(temp.name)) {
      result = temp.query;
      break;
      }
    }
  return result;
  }

  /**
   * This function is used to fill the combo boxes with the selection alternatives
   * if this operator is created as a part of a user defined operator then the
   * relation and condition combo boxes are filled with user defined operator
   * input items.
  **/
  private void loadDesign(){
    txtName.setText(name);
    for (int index = 0 ; index < test.window.designRelation.size(); index++) {
      cmbRelation1.addItem((String) test.window.designRelation.elementAt
                                (index));
      cmbRelation2.addItem((String) test.window.designRelation.elementAt
                                (index));
    }

    for (int opIndex = 0 ; opIndex < test.window.designOps.size(); opIndex++) {
      if (!name.equalsIgnoreCase((String) test.window.designOps.elementAt
                                      (opIndex))){
        cmbRelation1.addItem((String) test.window.designOps.elementAt
                                  (opIndex));
        cmbRelation2.addItem((String) test.window.designOps.elementAt
                                  (opIndex));
      }
    }

    for (int attIndex = 0; attIndex < test.window.designAttribute.size();
          attIndex++){
      cmbAttribute.addItem((String) test.window.designAttribute.elementAt
                                (attIndex));
    }
  }
}
```

14. OperatorEqjoin.Java

```
package ThesisGUI;

import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;

/**
 * This frame is used to allow the user to enter the operator information
 * for Eqjoin operator. It also allwos user to change the information
 * previously entered.
 * AUTHOR : IS & BA
**/
public class OperatorEqjoin extends JFrame implements WindowListener{
 /** Type of the operator (constant) **/
 final static String OPERATOR_TYPE = "Eqjoin";

 /** Input node number of the operator (constant) **/
 final static int NO_INPUT_NODES = 3;

 /** X coordinate of the operator**/
 private int x;

 /** Y coordinate of the operator**/
 private int y;

 /** name of the operator**/
 private String name;

 /** first input node relation**/
 private String relation1;

 /** second input node relation**/
 private String relation2;

 /** Join attribute list**/
 private String joinAtList;

 /** display information for the operator**/
 private boolean display = false;

 /** SQL query of the operator**/
 private String query;

 /** Specify why this frame is called (to create a new one (false)
   * or change the current operator (true)**/
 private boolean whoIsCalling = false;
```

```
/** The index of the operator in the Operators list **/
private int indexNum;

/** The index of the operator in the DFQLList**/
private int current;

/** name label of the frame**/
JLabel lblName          = new JLabel();

/** first relation label of the frame**/
JLabel lblRelation1     = new JLabel();

/** second relation label of the frame**/
JLabel lblRelation2     = new JLabel();

/** text area to enter the name of the operator default diff#**/
JTextField txtName      = new JTextField();

/** Okay button to confirm the info entered**/
JButton btnOK           = new JButton();

/** check box for display information**/
JCheckBox chkResult     = new JCheckBox();

/** Combo box lists the available operators or table names that can be
  * related to this operator**/
JComboBox cmbRelation1  = new JComboBox();

/** Combo box lists the available operators or table names that can be
  * related to this operator**/
JComboBox cmbRelation2  = new JComboBox();

/** join attribute list label of the frame**/
JLabel lblJoinAtList    = new JLabel();

/** text area to enter the join attribute list for the operator**/
JTextField txtJoinAtList = new JTextField();

/** Combo box lists the available conditions that can be
  * related to this operator**/
JComboBox cmbAttribute  = new JComboBox();

/** Button cancel to cancel this operation**/
JButton btnCancel       = new JButton();
```

```
/*              CONSTRUCTOR                    */
/**
 * Default constructor
 * @parameters X and Y (integer) coordinates of the operator
**/
public OperatorEqjoin(int x, int y) {
 try  {
  this.x = x;
  this.y = y;
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}

/**
 * Constructor with all parameters
 * @parameters @ parameters x, y, caller1, caller2 (integer),
 * name,relation1, relation2, joinAtList        (String),
 * display                                      (boolean)
**/
public OperatorEqjoin(int x, int y, int caller1, int caller2, String name,
                String relation1, String relation2, String joinAtList,
                boolean display){
 this.x              = x;
 this.y              = y;
 this.current        = caller1;
 this.indexNum       = caller2;
 this.whoIsCalling   = true;
 this.name           = name;
 this.relation1      = relation1;
 this.relation2      = relation2;
 this.joinAtList     = joinAtList;
 this.display        = display;

 txtName.setText      (name);
 txtJoinAtList.setText (joinAtList);

 if (display){
  chkResult.setSelected(true);
 }
 else{
  chkResult.setSelected(false);
 }
```

```
  try {
   jbInit();
  }
   catch(Exception e) {
    e.printStackTrace();
  }
 }

/**
  * Design constructor
  * @parameters name
**/
public OperatorEqjoin(String name) {
  try {
   this.name = name;
   jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
 }

/*              INITIATE METHOD                  */
/**
  * The initiation
  * creates the frame for user to enter the operator data, and adds
  * buttons, labels, text area and combo boxes.
**/
private void jbInit() throws Exception {
 lblName.setText              ("Name");
 this.setTitle                ("Eqjoin");
 lblRelation1.setText         ("Relation1");
 lblRelation2.setText         ("Relation2");
 lblJoinAtList.setText        ("JoinAttList");
 btnOK.setText                ("OK");
 chkResult.setText            ("Display");
 btnCancel.setText            ("Cancel");

 lblName.setBounds            (new Rectangle(4, 2, 93, 18));
 lblRelation1.setBounds       (new Rectangle(4, 27, 93, 18));
 lblRelation2.setBounds       (new Rectangle(4, 49, 93, 18));
 lblJoinAtList.setBounds      (new Rectangle(4, 72, 93, 18));
 txtName.setBounds            (new Rectangle(80, 2, 114, 17));
 txtJoinAtList.setBounds      (new Rectangle(80, 73, 114, 17));
 cmbAttribute.setBounds       (new Rectangle(80, 73, 114, 17));
 btnOK.setBounds              (new Rectangle(142, 95, 53, 22));
```

```
chkResult.setBounds          (new Rectangle(2, 95, 63, 22));
cmbRelation1.setBounds     (new Rectangle(80, 28, 114, 17));
cmbRelation2.setBounds     (new Rectangle(80, 50, 114, 17));
btnCancel.setBounds          (new Rectangle(70, 95, 68, 22));

if (test.window.isDesign()){
 loadDesign();
}
else {
 loadSelect();
}

btnOK.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnOK_actionPerformed(e);
 }});

btnCancel.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnCancel_actionPerformed(e);
 }});

this.getContentPane().setLayout (null);
cmbRelation1.setBorder            (null);
cmbRelation2.setBorder            (null);
chkResult.setBorderPainted        (true);

btnOK.setFont      (new java.awt.Font("Dialog", 0, 11));
chkResult.setFont (new java.awt.Font("Dialog", 0, 11));
btnCancel.setFont (new java.awt.Font("Dialog", 0, 11));

btnOK.setBorder      (BorderFactory.createRaisedBevelBorder());
chkResult.setBorder (BorderFactory.createRaisedBevelBorder());
btnCancel.setBorder (BorderFactory.createRaisedBevelBorder());

this.getContentPane().add(lblName, null);
this.getContentPane().add(txtName, null);
this.getContentPane().add(lblRelation1, null);
this.getContentPane().add(cmbRelation1, null);
this.getContentPane().add(cmbRelation2, null);
this.getContentPane().add(lblRelation2, null);

if (test.window.isDesign()){
 this.getContentPane().add(cmbAttribute, null);
}
else{
    this.getContentPane().add(txtJoinAtList, null);
   }
  this.getContentPane().add(lblJoinAtList, null);
  this.getContentPane().add(chkResult, null);
  this.getContentPane().add(btnCancel, null);
  this.getContentPane().add(btnOK, null);

  this.setBounds(100,100,210,150);
  this.show(true);
 }

/*              Set Methods                  */
/**
  * Sets the first relation of the operator
  * @parameter relation1 (String)
**/
public void setRelation1 (String relation1){
 this.relation1 = relation1;
}

/**
  * Sets the second relation of the operator
  * @parameter relation2 (String)
**/
public void setRelation2 (String relation2){
 this.relation2 = relation2;
}

/**
  * Sets the name of the operator
  * @parameter name (String)
**/
public void setName (String name){
 this.name = name;
}

/**
  * Sets the join attribute list of the operator
  * @parameter joinAtList (String)
**/
public void setJoinAtList (String joinAtList){
 this.joinAtList = joinAtList;
}
```

```
/*              Get Methods                     */
/**
 * Gets the number of the input nodes of the operator
**/
public int getnumberOfNodes(){
 return this.NO_INPUT_NODES;
}

/**
 * Gets the type of the operator
**/
public String getOperatorType(){
 return this.OPERATOR_TYPE;
}

/**
 * Gets the first relationof the operator
**/
public String getRelation1 (){
 return this.relation1;
}

/**
 * Gets the second relation of the operator
**/
public String getRelation2 (){
 return this.relation2;
}

/**
 * Gets the name of the operator
**/
public String getName (){
 return this.name;
}

/**
 * Gets the join attribute list of the operator
**/
public String getJoinAtList (){
 return this.joinAtList;
}
```

```
/*           WINDOW ACTIVATIONS              */
/**
 * The window listener methods
 * @param event (WindowEvent)
**/
public void windowClosing        (WindowEvent event){System.exit(0);}
public void windowActivated      (WindowEvent event){}
public void windowClosed         (WindowEvent event){}
public void windowDeactivated    (WindowEvent event){}
public void windowDeiconified    (WindowEvent event){}
public void windowIconified      (WindowEvent event){}
public void windowOpened         (WindowEvent event){}

/**
 * button okay action performed
 * Takes the information entered for the operator. Creates a new operator and
 * calls add or change method of the draw class. And sends new operator
 * information there to be added to the list.
 * @param e (ActionEvent)
**/
void btnOK_actionPerformed(ActionEvent e) {
 if(chkResult.isSelected()){
    display = true;
 }
 else{
  display = false;
 }
 setName     (txtName.getText());
 setRelation1((String)cmbRelation1.getSelectedItem());
 setRelation2((String)cmbRelation2.getSelectedItem());

 if (test.window.isDesign()){
   setJoinAtList((String)cmbAttribute.getSelectedItem());
 }
 else {
  if (txtJoinAtList.getText().equals("")){
    test.window.message.setMessage("You did not enter the Eqjoin attribute");
    test.window.message.show(true);
  }
  else {
    setJoinAtList(txtJoinAtList.getText());
  }
 }

 String [] input = {getRelation1(),getRelation2(),getJoinAtList(),"none"};
 buildQuery();
 if (test.window.isDesign()){
  test.window.designQueries.addElement(query);
 }
 else {
  if (whoIsCalling){
   test.window.cnvDFQL.change
                        (current,indexNum,x,y,getName(),getnumberOfNodes(),
                         input,display, getOperatorType(),query);
  }
  else{
   test.window.cnvDFQL.add
                        (x,y,getName(),getnumberOfNodes(),input,display,
                         getOperatorType(), query);
  }
 }
 this.dispose();
}

/**
 * Loads the relations with the tables of the selected database and previously
 * created operators.
**/
void loadSelect() {

 if(!whoIsCalling){
  txtName.setText("Eqjoin"+test.window.Counter[6]++);
 }

 Vector deneme = test.window.currentDatabase.getTableNames();
 for (int index = 0 ; index < deneme.size(); index++) {
   cmbRelation1.addItem((String) deneme.elementAt(index));
   cmbRelation2.addItem((String) deneme.elementAt(index));
 }

 for (int xi = 0; xi <test.window.cnvDFQL.numOfOperators;xi++){
  cmbRelation1.addItem(test.window.cnvDFQL.operators[xi].name);
  cmbRelation2.addItem(test.window.cnvDFQL.operators[xi].name);
 }
 cmbRelation1.removeItem(name);
 cmbRelation2.removeItem(name);

 if (relation1== null)
  cmbRelation1.setSelectedIndex(0);
 else
  cmbRelation1.setSelectedItem(relation1);
 if (relation2== null)
   cmbRelation2.setSelectedIndex(0);
 else
   cmbRelation2.setSelectedItem(relation2);
}

/**
  * button cancel action performed
  * terminates this frame without creating an operator. decrements the counter
  * @param e (ActionEvent)
**/
void btnCancel_actionPerformed(ActionEvent e) {
 test.window.Counter[6]--;
 this.dispose();
}

/**
 * This portion of the code produces the SQL query for this operator according
 * to the information entered by user.
**/
public void buildQuery() {

 String checkRelation = isOtherOperator (relation1);
 Vector attList2 = new Vector();
 String queryPre = "select distinct s1.*";
 if (checkRelation == null) {
    query = " from " + relation1+" s1,";
 }
 else {
    query = " from " + "(" +checkRelation + ") s1,";
 }

 checkRelation = isOtherOperator(relation2);
 try {
  if (checkRelation == null) {
     query = query + relation2+ " s2" ;
     attList2 = test.window.currentDatabase.getTableAttributeNames(relation2);
  }
  else {
    query = query + "(" +checkRelation + ") s2";
    Table temp = test.window.currentDatabase.executeQuery(checkRelation);
    attList2 = temp.getAttributeNames();
  }

  if (!(joinAtList.equals(""))) {
    query = query + " where s1." + joinAtList+" = s2."+joinAtList;
```

```
      }

      int deneme =  attList2.indexOf(joinAtList);
      if (deneme >= 0){
       for (int dup = 0; dup<attList2.size(); dup++){
         if (dup != deneme){
          queryPre = queryPre + ", s2."+(String)attList2.elementAt(dup);
         }
       }
      }
      else {
       queryPre = queryPre + ", s2.*";
      }
      query = queryPre + query;
    }
    catch (Exception e){
     System.out.println(e.getMessage());
    }
  }

  /**
   * This portion of the code looks for the type of the relations, if they are
   * tables it returns null, otherwise if is another operator, it returns the
   * query of the operator which the current operator related to.
  **/
  public String isOtherOperator (String type) {
    String result = null;
    for (int index = 0; index < test.window.DFQLQuery.size(); index++) {
      Operator temp = (Operator) test.window.DFQLQuery.elementAt(index);
      if (type.equalsIgnoreCase(temp.name)) {
        result = temp.query;
        break;
      }
    }
  return result;
  }

   /**
   * This function is used to fill the combo boxes with the selection alternatives
   * if this operator is created as a part of a user defined operator then the
   * relation and condition combo boxes are filled with user defined operator
   * input items.
  **/
  private void loadDesign(){
   txtName.setText(name);
   for (int index = 0 ; index < test.window.designRelation.size(); index++) {
    cmbRelation1.addItem((String) test.window.designRelation.elementAt
                                  (index));
    cmbRelation2.addItem((String) test.window.designRelation.elementAt
                                  (index));
   }

   for (int opIndex = 0 ; opIndex < test.window.designOps.size(); opIndex++) {
    if (!name.equalsIgnoreCase((String) test.window.designOps.elementAt
                                        (opIndex))){
      cmbRelation1.addItem((String) test.window.designOps.elementAt
                                    (opIndex));
      cmbRelation2.addItem((String) test.window.designOps.elementAt
                                    (opIndex));
    }}

   for (int attIndex = 0; attIndex < test.window.designAttribute.size();
        attIndex++){
     cmbAttribute.addItem((String) test.window.designAttribute.elementAt
                                   (attIndex));
   }
  }
}
```

15. OperatorGroupCnt.Java

```
package ThesisGUI;

import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;

/**
 * This frame is used to allow the user to enter the operator information
 * for group count operator. It also allwos user to change the information
 * previously entered.
 * AUTHOR : IS & BA
**/
public class OperatorGroupCnt extends JFrame implements WindowListener{

 /** Type of the operator (constant) **/
 final static String OPERATOR_TYPE = "GroupCnt";
```

```
/** Input node number of the operator (constant) **/
final static int NO_INPUT_NODES = 3;

/** X coordinate of the operator**/
private int x;

/** Y coordinate of the operator**/
private int y;

/** name of the operator**/
private String name;

/** input node relation**/
private String relation;

/** group attribute list**/
private String groupAtt;

/** count attribute list**/
private String countAtt;

/** display information for the operator**/
private boolean display = false;

/** SQL query of the operator**/
private String query;

/** Specify why this frame is called (to create a new one (false)
  * or change the current operator (true)**/
private boolean whoIsCalling = false;

/** The index of the operator in the Operators list **/
private int indexNum;

/** The index of the operator in the DFQLList**/
private int current;

/** name label of the frame**/
JLabel lblName          = new JLabel();

/** relation label of the frame**/
JLabel lblRelation      = new JLabel();

/** group attribute list label of the frame**/
JLabel lblGroupAtt  = new JLabel();
```

```
/** count attribute list label of the frame**/
JLabel lblCountAtt  = new JLabel();

/** text area to enter the name of the operator default diff#**/
JTextField txtName      = new JTextField();

/** Okay button to confirm the info entered**/
JButton btnOK          = new JButton();

/** check box for display information**/
JCheckBox chkResult     = new JCheckBox();

/** Combo box lists the available operators or table names that can be
  * related to this operator**/
JComboBox cmbRelation1  = new JComboBox();

/** Button cancel to cancel this operation**/
JButton btnCancel       = new JButton();

/** text area to enter the group attribute list of the operator **/
JTextField txtGroupAtt  = new JTextField();

/** Combo box lists the available attributes that can be
  * related to this operator**/
JComboBox cmbGrpAttribute  = new JComboBox();

/** text area to enter the count attribute list of the operator **/
JTextField txtCountAtt = new JTextField();

/** Combo box lists the available attributes that can be
  * related to this operator**/
JComboBox cmbCntAttribute  = new JComboBox();

/*             CONSTRUCTOR                    */
/**
  * Default constructor
  * @parameters X and Y (integer) coordinates of the operator
**/
public OperatorGroupCnt(int x, int y) {
  try  {
    this.x = x;
    this.y = y;
    jbInit();
  }
  catch(Exception e) {
    e.printStackTrace();
   }
 }

/**
  * Constructor with all parameters
  * @parameters @ parameters x, y, caller1, caller2 (integer),
  * name,relation, groupAtt, countAtt            (String),
  * display                                    (boolean)
**/
public OperatorGroupCnt(int x, int y, int caller1, int caller2, String name,
                String relation, String groupAtt, String countAtt,
                boolean display){
  this.x              = x;
  this.y              = y;
  this.current        = caller1;
  this.indexNum       = caller2;
  this.whoIsCalling   = true;
  this.name           = name;
  this.relation       = relation;
  this.groupAtt       = groupAtt;
  this.countAtt       = countAtt;
  this.display        = display;

  txtName.setText      (name);
  txtGroupAtt.setText (groupAtt);
  txtCountAtt.setText (countAtt);
  if (display){
   chkResult.setSelected(true);
  }
  else{
   chkResult.setSelected(false);
  }
  try {
   jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
}
```

```
/**
 * Design constructor
 * @parameters name
**/
public OperatorGroupCnt(String name) {
 try {
  this.name = name;
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}

/*              INITIATE METHOD                    */
/**
 * The initiation
 * creates the frame for user to enter the operator data, and adds
 * buttons, labels, text area and combo boxes.
**/
private void jbInit() throws Exception {
 lblName.setText              ("Name");
 this.setTitle                ("GroupCnt");
 lblRelation.setText          ("Relation");
 lblGroupAtt.setText          ("Group Att.");
 lblCountAtt.setText          ("Count Att.");
 btnOK.setText                ("OK");
 chkResult.setText            ("Display");
 btnCancel.setText            ("Cancel");

 lblName.setBounds                 (new Rectangle(4, 2, 93, 18));
 lblRelation.setBounds             (new Rectangle(4, 27, 93, 18));
 lblGroupAtt.setBounds             (new Rectangle(4, 52, 93, 18));
 lblCountAtt.setBounds             (new Rectangle(4, 75, 93, 18));
 txtName.setBounds                 (new Rectangle(80, 2, 114, 17));
 txtGroupAtt.setBounds             (new Rectangle(80, 53, 114, 17));
 cmbGrpAttribute.setBounds         (new Rectangle(80, 53, 114, 17));
 txtCountAtt.setBounds             (new Rectangle(80, 76, 114, 17));
 cmbCntAttribute.setBounds         (new Rectangle(80, 76, 114, 17));
 btnOK.setBounds                   (new Rectangle(144, 101, 49, 22));
 chkResult.setBounds               (new Rectangle(4, 101, 60, 22));
 cmbRelation1.setBounds            (new Rectangle(80, 28, 114, 17));
 btnCancel.setBounds               (new Rectangle(74, 101, 60, 23));

 if (test.window.isDesign()){
  loadDesign();
}
else{ loadSelect();
}

btnOK.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnOK_actionPerformed(e);
 }});

btnCancel.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnCancel_actionPerformed(e);
 }});

btnOK.setBorder          (BorderFactory.createRaisedBevelBorder());
btnCancel.setBorder      (BorderFactory.createRaisedBevelBorder());
chkResult.setBorder      (BorderFactory.createRaisedBevelBorder());

chkResult.setBorderPainted     (true);
cmbRelation1.setBorder         (null);

btnCancel.setFont  (new java.awt.Font("Dialog", 0, 11));
btnOK.setFont      (new java.awt.Font("Dialog", 0, 11));
chkResult.setFont  (new java.awt.Font("Dialog", 0, 11));

this.getContentPane().setLayout(null);
this.getContentPane().add(lblName, null);
this.getContentPane().add(txtName, null);
this.getContentPane().add(lblRelation, null);
this.getContentPane().add(cmbRelation1, null);
if (test.window.isDesign()){
  this.getContentPane().add(cmbGrpAttribute,null);
  this.getContentPane().add(cmbCntAttribute, null);
}
else {
  this.getContentPane().add(txtGroupAtt, null);
  this.getContentPane().add(txtCountAtt, null);
}
this.getContentPane().add(lblGroupAtt, null);
this.getContentPane().add(lblCountAtt, null);
this.getContentPane().add(chkResult, null);
this.getContentPane().add(btnCancel, null);
this.getContentPane().add(btnOK, null);

this.setBounds(100,100,210,160);
this.show(true);}
```

```
/*              Set Methods                 */
/**
  * Sets the relation of the operator
  * @parameter relation (String)
**/
public void setRelation (String relation){
 this.relation = relation;
}

/**
  * Sets the group attribute list
  * @parameter groupAtt (String)
**/
public void setGroupAtt (String groupAtt){
 this.groupAtt = groupAtt;
}

/**
  * Sets the count attribute list
  * @parameter countAtt (String)
**/
public void setCountAtt (String countAtt){
 this.countAtt = countAtt;
}

/**
  * Sets the name of the operator
  * @parameter name (String)
**/
public void setName (String name){
 this.name = name;
}

/**
  * gets the input node number of the operator
**/
public int getnumberOfNodes(){
 return this.NO_INPUT_NODES;
}

/**
  * gets the operator type of the operator
**/
public String getOperatorType(){
 return this.OPERATOR_TYPE;}
```

```
/**
  * gets the relation of the operator
**/
public String getRelation (){
  return this.relation;
}

/**
  * gets the group attribute list of the operator
**/
public String getGroupAtt (){
  return this.groupAtt;
}

/**
  * gets the count attribute list of the operator
**/
public String getCountAtt (){
  return this.countAtt;
}

/**
  * gets the name of the operator
**/
public String getName (){
  return this.name;
}

/*              WINDOW ACTIVATIONS                    */
/**
  * The window listener methods
  * @param event (WindowEvent)
**/
public void windowClosing         (WindowEvent event){System.exit(0);}
public void windowActivated       (WindowEvent event){ }
public void windowClosed          (WindowEvent event){ }
public void windowDeactivated     (WindowEvent event){ }
public void windowDeiconified     (WindowEvent event){ }
public void windowIconified       (WindowEvent event){ }
public void windowOpened          (WindowEvent event){ }
```

```
/**
 * button okay action performed
 * Takes the information entered for the operator. Creates a new operator and
 * calls add or change method of the draw class. And sends new operator
 * information there to be added to the list.
 * @param e (ActionEvent)
**/
void btnOK_actionPerformed(ActionEvent e) {
 if(chkResult.isSelected()){
    display = true;
 }
 else{
   display = false;
 }
 setName(txtName.getText());
 setRelation((String)cmbRelation1.getSelectedItem());

 if (test.window.isDesign()){
   setGroupAtt((String)cmbGrpAttribute.getSelectedItem());
   setCountAtt((String)cmbCntAttribute.getSelectedItem());
 }
 else{
   setGroupAtt(txtGroupAtt.getText());
   setCountAtt(txtCountAtt.getText());
   if (txtCountAtt.getText().equals("")){
      test.window.message.setMessage("You did not enter the count attribute");
      test.window.message.show(true);
   }
 }

 String [] input = {getRelation(),getGroupAtt(),getCountAtt(),"none"};

 buildQuery();

 if (test.window.isDesign()){
   test.window.designQueries.addElement(query);
 }
 else {
   if (whoIsCalling){
     test.window.cnvDFQL.change
                         (current,indexNum,x,y,getName(),getnumberOfNodes(),
                          input,display, getOperatorType(),query);
   }
```

```
    else{
      test.window.cnvDFQL.add
                          (x,y,getName(),getnumberOfNodes(),input,display,
                           getOperatorType(), query);
    }
   }
  this.dispose();
 }

 /**
  * Loads the relations with the tables of the selected database and previously
  * created operators.
 **/
 void loadSelect() {
  Vector deneme = test.window.currentDatabase.getTableNames();

  if(!whoIsCalling){
   txtName.setText("GroupCnt"+test.window.Counter[5]++);
  }

  for (int index = 0 ; index < deneme.size(); index++) {
    cmbRelation1.addItem((String) deneme.elementAt(index));
  }

  for (int xi = 0; xi <test.window.cnvDFQL.numOfOperators;xi++){
   cmbRelation1.addItem(test.window.cnvDFQL.operators[xi].name);
  }
  cmbRelation1.removeItem(name);

  if (relation== null)
    cmbRelation1.setSelectedIndex(0);
  else
    cmbRelation1.setSelectedItem(relation);
 }

 /**
  * button cancel action performed
  * terminates this frame without creating an operator. decrements the counter
  * @param e (ActionEvent)
 **/
 void btnCancel_actionPerformed(ActionEvent e) {
   test.window.Counter[5]--;
   this.dispose();
 }
```

```
/**
 * This portion of the code produces the SQL query for this operator according
 * to the information entered by user.
**/
public void buildQuery(){
  String checkRelation = isOtherOperator (relation);
  if (countAtt.equals("")){
    query = "select distinct " + groupAtt + ", count (*)";
  }
  else{
    query = "select distinct " + groupAtt + ", count (*) As " + countAtt;
  }

  if (checkRelation == null) {
    query = query + " from " + relation;
  }
  else {
    query = query + " from " + "(" +checkRelation + ")";
  }

  query = query + " group by " + groupAtt;
}

/**
 * This portion of the code looks for the type of the relations, if they are
 * tables it returns null, otherwise if is another operator, it returns the
 * query of the operator which the current operator related to.
**/
public String isOtherOperator (String type) {
  String result = null;
  for (int index = 0; index < test.window.DFQLQuery.size(); index++) {
    Operator temp = (Operator) test.window.DFQLQuery.elementAt(index);
    if (type.equalsIgnoreCase(temp.name)) {
      result = temp.query;
      break;
    }
  }
  return result;
}

/**
 * This function is used to fill the combo boxes with the selection alternatives
 * if this operator is created as a part of a user defined operator then the
 * relation and condition combo boxes are filled with user defined operator
 * input items.
**/
```

```
    private void loadDesign(){
      txtName.setText(name);
      for (int index = 0 ; index < test.window.designRelation.size(); index++) {
        cmbRelation1.addItem((String) test.window.designRelation.elementAt
                                   (index));
      }

      for (int opIndex = 0 ; opIndex < test.window.designOps.size(); opIndex++) {
        if (!name.equalsIgnoreCase((String) test.window.designOps.elementAt
                                       (opIndex)))
          cmbRelation1.addItem((String) test.window.designOps.elementAt
                                    (opIndex));
      }

      for (int attIndex = 0; attIndex < test.window.designAttribute.size();
            attIndex++){
        cmbGrpAttribute.addItem((String) test.window.designAttribute.elementAt
                                       (attIndex));
        cmbCntAttribute.addItem((String) test.window.designAttribute.elementAt
                                       (attIndex));
      }
    }
  }
```

16. **OperatorGrpAllSat.Java**

```
package ThesisGUI;

import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;

/**
  * This frame is used to allow the user to enter the operator information
  * for group all satisfy operator. It also allwos user to change the information
  * previously entered.
  * AUTHOR : IS & BA
**/
public class OperatorGrpAllSat extends JFrame implements WindowListener{
```

```
/** Type of the operator (constant) **/
final static String OPERATOR_TYPE = "GrpAllSat";

/** Input node number of the operator (constant) **/
final static int NO_INPUT_NODES = 3;

/** X coordinate of the operator**/
private int x;

/** Y coordinate of the operator**/
private int y;

/** name of the operator**/
private String name;

/** input node relation**/
private String relation;

/** group attribute list**/
private String groupAtt;

/** groupping condition **/
private String condition;

/** display information for the operator**/
private boolean display = false;

/** SQL query of the operator**/
private String query;

/** Specify why this frame is called (to create a new one (false)
  * or change the current operator (true)**/
private boolean whoIsCalling = false;

/** The index of the operator in the Operators list **/
private int indexNum;

/** The index of the operator in the DFQLList**/
private int current;

/** name label of the frame**/
JLabel lblName          = new JLabel();

/** relation label of the frame**/
JLabel lblRelation      = new JLabel();
```

```
/** group attribute list label of the frame**/
JLabel lblGroupAtt  = new JLabel();

/** count attribute list label of the frame**/
JLabel lblCondition   = new JLabel();

/** text area to enter the name of the operator default diff#**/
JTextField txtName      = new JTextField();

/** Okay button to confirm the info entered**/
JButton btnOK          = new JButton();

/** check box for display information**/
JCheckBox chkResult     = new JCheckBox();

/** Combo box lists the available operators or table names that can be
  * related to this operator**/
JComboBox cmbRelation1  = new JComboBox();

/** Button cancel to cancel this operation**/
JButton btnCancel       = new JButton();

/** text area to enter the group attribute list of the operator **/
JTextField txtGroupAtt = new JTextField();

/** Combo box lists the available attributes that can be
  * related to this operator**/
JComboBox cmbGroupAtt  = new JComboBox();

/** text area to enter the count attribute list of the operator **/
JTextField txtCondition = new JTextField();

/** Combo box lists the available attributes that can be
  * related to this operator**/
JComboBox cmbCondition = new JComboBox();

/*              CONSTRUCTOR                   */
/**
  * Default constructor
  * @parameters X and Y (integer) coordinates of the operator
**/
public OperatorGrpAllSat(int x, int y) {
 try {
  this.x = x;
  this.y = y;
  jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
 }

/**
  * Constructor with all parameters
  * @parameters @ parameters x, y, caller1, caller2 (integer),
  * name,relation, groupAtt, condition            (String),
  * display                                        (boolean)
**/
public OperatorGrpAllSat(int x, int y, int caller1, int caller2, String name,
                    String relation, String groupAtt, String condition,
                    boolean display){
  this.x              = x;
  this.y              = y;
  this.current        = caller1;
  this.indexNum       = caller2;
  this.whoIsCalling   = true;
  this.name           = name;
  this.relation       = relation;
  this.groupAtt       = groupAtt;
  this.condition      = condition;
  this.display        = display;

  txtName.setText            (name);
  txtGroupAtt.setText        (groupAtt);
  txtCondition.setText       (condition);

  if (display){
   chkResult.setSelected(true);
  }
  else{
   chkResult.setSelected(false);
  }

  try {
   jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
}
```

```
/**
 * Design constructor
 * @parameters name
**/
public OperatorGrpAllSat(String name) {
 try {
  this.name = name;
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}

/*              INITIATE METHOD                    */
/**
 * The initiation
 * creates the frame for user to enter the operator data, and adds
 * buttons, labels, text area and combo boxes.
**/
private void jbInit() throws Exception {
 lblName.setText              ("Name");
 this.setTitle                ("GrpAllSat");
 lblRelation.setText          ("Relation");
 lblGroupAtt.setText          ("Group Att.");
 lblCondition.setText         ("Condition");
 btnOK.setText                ("OK");
 chkResult.setText            ("Display");
 btnCancel.setText            ("Cancel");

 lblName.setBounds            (new Rectangle(4, 2, 93, 18));
 lblRelation.setBounds        (new Rectangle(4, 27, 93, 18));
 lblGroupAtt.setBounds        (new Rectangle(3, 53, 93, 18));
 lblCondition.setBounds       (new Rectangle(3, 76, 93, 18));
 txtName.setBounds            (new Rectangle(80, 2, 114, 17));
 txtGroupAtt.setBounds        (new Rectangle(79, 54, 114, 17));
 cmbGroupAtt.setBounds        (new Rectangle(79, 54, 114, 17));
 txtCondition.setBounds       (new Rectangle(79, 77, 114, 17));
 cmbCondition.setBounds       (new Rectangle(79, 77, 114, 17));
 btnOK.setBounds              (new Rectangle(143, 104, 50, 22));
 chkResult.setBounds          (new Rectangle(3, 104, 62, 22));
 cmbRelation1.setBounds       (new Rectangle(80, 28, 114, 17));
 btnCancel.setBounds          (new Rectangle(74, 104, 60, 23));

 if (test.window.isDesign()){
  loadDesign();
}
else{
 loadSelect();
}

btnOK.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnOK_actionPerformed(e);
 }});

btnCancel.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnCancel_actionPerformed(e);
 }});

chkResult.setFont   (new java.awt.Font("Dialog", 0, 11));
btnOK.setFont       (new java.awt.Font("Dialog", 0, 11));
btnCancel.setFont   (new java.awt.Font("Dialog", 0, 11));

btnCancel.setBorder        (BorderFactory.createRaisedBevelBorder());
btnOK.setBorder            (BorderFactory.createRaisedBevelBorder());
chkResult.setBorder        (BorderFactory.createRaisedBevelBorder());

chkResult.setBorderPainted          (true);
cmbRelation1.setBorder              (null);
this.getContentPane().setLayout     (null);

this.getContentPane().add(lblName, null);
this.getContentPane().add(txtName, null);
this.getContentPane().add(lblRelation, null);
this.getContentPane().add(cmbRelation1, null);
this.getContentPane().add(lblGroupAtt, null);
this.getContentPane().add(lblCondition, null);
if(test.window.isDesign()){
 this.getContentPane().add(cmbGroupAtt,null);
 this.getContentPane().add(cmbCondition,null);
}
else{
 this.getContentPane().add(txtGroupAtt, null);
 this.getContentPane().add(txtCondition, null);
}
this.getContentPane().add(chkResult, null);
this.getContentPane().add(btnCancel, null);
this.getContentPane().add(btnOK, null);

this.setBounds(100,100,210,160);
```

```
  this.show(true);
 }

 /*             Set Methods                  */
 /**
   * Sets the relation of the operator
   * @parameter relation (String)
 **/
 public void setRelation (String relation){
  this.relation = relation;
 }

 /**
   * Sets the group attribute list of the operator
   * @parameter groupAtt (String)
 **/
 public void setGroupAtt (String groupAtt){
  this.groupAtt = groupAtt;
 }

 /**
   * Sets the condition of the operator
   * @parameter condition (String)
 **/
 public void setCondition (String condition){
  this.condition = condition;
 }

 /**
   * Sets the name of the operator
   * @parameter name (String)
 **/
 public void setName (String name){
  this.name = name;
 }

 /*             Get Methods                  */
 /**
   * gets the input number of the operator
 **/
 public int getnumberOfNodes(){
  return this.NO_INPUT_NODES;
 }
```

```
/**
  * gets the type of the operator
**/
public String getOperatorType(){
  return this.OPERATOR_TYPE;
}

/**
  * gets the relation of the operator
**/
public String getRelation (){
  return this.relation;
}

/**
  * gets the group Attribute list of the operator
**/
public String getGroupAtt (){
  return this.groupAtt;
}

/**
  * gets the condition of the operator
**/
public String getCondition (){
  return this.condition;
}

/**
  * gets the name of the operator
**/
public String getName (){
  return this.name;
}

/*              WINDOW ACTIVATIONS                    */
/**
  * The window listener methods
  * @param event (WindowEvent)
**/
public void windowClosing         (WindowEvent event){System.exit(0);}
public void windowActivated       (WindowEvent event){}
public void windowClosed          (WindowEvent event){}
public void windowDeactivated     (WindowEvent event){}
public void windowDeiconified     (WindowEvent event){}
public void windowIconified       (WindowEvent event){}
public void windowOpened          (WindowEvent event){}

/**
 * button okay action performed
 * Takes the information entered for the operator. Creates a new operator and
 * calls add or change method of the draw class. And sends new operator
 * information there to be added to the list.
 * @param e (ActionEvent)
**/
void btnOK_actionPerformed(ActionEvent e) {
 if(chkResult.isSelected()){
    display = true;
 }
 else{
   display = false;
 }
 setName(txtName.getText());

 setRelation((String)cmbRelation1.getSelectedItem());
 if (test.window.isDesign()){
   setGroupAtt((String)cmbGroupAtt.getSelectedItem());
   setCondition((String)cmbCondition.getSelectedItem());
 }
 else{
    setGroupAtt(txtGroupAtt.getText());
    if (txtCondition.getText().equals("")){
      test.window.message.setMessage("You did not enter the condition");
      test.window.message.show(true);
    }
    setCondition(txtCondition.getText());
 }
String [] input = {getRelation(),getGroupAtt(),getCondition(),"none"};
buildQuery();

if (test.window.isDesign()){
  test.window.designQueries.addElement(query);
}
else {
  if (whoIsCalling){
     test.window.cnvDFQL.change
                        (current,indexNum,x,y,getName(),getnumberOfNodes(),
                         input,display, getOperatorType(),query);
  }
  else{
     test.window.cnvDFQL.add
                        (x,y,getName(),getnumberOfNodes(),input,display,
                        getOperatorType(), query);
   }
  }
  this.dispose();
 }

/**
  * Loads the relations with the tables of the selected database and previously
  * created operators.
**/
void loadSelect() {
  Vector deneme = test.window.currentDatabase.getTableNames();

  if(!whoIsCalling){
   txtName.setText("GrpAllSat"+test.window.Counter[7]++);
  }

  for (int index = 0 ; index < deneme.size(); index++) {
    cmbRelation1.addItem((String) deneme.elementAt(index));
  }

  for (int xi = 0; xi <test.window.cnvDFQL.numOfOperators;xi++){
   cmbRelation1.addItem(test.window.cnvDFQL.operators[xi].name);
  }
  cmbRelation1.removeItem(name);

  if (relation== null)
    cmbRelation1.setSelectedIndex(0);
  else
    cmbRelation1.setSelectedItem(relation);
}

/**
  * button cancel action performed
  * terminates this frame without creating an operator. decrements the counter
  * @param e (ActionEvent)
**/

void btnCancel_actionPerformed(ActionEvent e) {
   test.window.Counter[7]--;
   this.dispose();
}
```

```
/**
  * This portion of the code produces the SQL query for this operator according
  * to the information entered by user.
**/
public void buildQuery(){
  String checkRelation = isOtherOperator (relation);
  query = "select distinct " + groupAtt;

  if (checkRelation == null) {
     query = query + " from " + relation;
  }
  else {
     query = query + " from " + "(" +checkRelation + ")";
  }

  if (!condition.equals("")){
   query = query + " where " + condition;
  }

  query =  query + " group by " + groupAtt;
}

/**
  * This portion of the code looks for the type of the relations, if they are
  * tables it returns null, otherwise if is another operator, it returns the
  * query of the operator which the current operator related to.
**/
public String isOtherOperator (String type) {
  String result = null;
  for (int index = 0; index < test.window.DFQLQuery.size(); index++) {
    Operator temp = (Operator) test.window.DFQLQuery.elementAt(index);
    if (type.equalsIgnoreCase(temp.name)) {
      result = temp.query;
      break;
    }
  }
  return result;
}

/**
  * This function is used to fill the combo boxes with the selection alternatives
  * if this operator is created as a part of a user defined operator then the
  * relation and condition combo boxes are filled with user defined operator
  * input items.
**/
private void loadDesign(){
    txtName.setText(name);
    for (int index = 0 ; index < test.window.designRelation.size(); index++) {
      cmbRelation1.addItem((String) test.window.designRelation.elementAt
                                    (index));
    }

    for (int opIndex = 0 ; opIndex < test.window.designOps.size(); opIndex++) {
      if (!name.equalsIgnoreCase((String) test.window.designOps.elementAt
                                        (opIndex)))
        cmbRelation1.addItem((String) test.window.designOps.elementAt
                                      (opIndex));
    }

    for (int conIndex = 0; conIndex < test.window.designCondition.size();
          conIndex++){
      cmbCondition.addItem((String) test.window.designCondition.elementAt
                                    (conIndex));
    }

    for (int attIndex = 0; attIndex < test.window.designAttribute.size();
                    attIndex++){
      cmbGroupAtt.addItem((String) test.window.designAttribute.elementAt
                                   (attIndex));
    }
  }
}
```

17. OperatorGrpAvg.Java

```
package ThesisGUI;

import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;

/**
  * This frame is used to allow the user to enter the operator information
  * for Group Average operator. It also allwos user to change the information
  * previously entered.
  * AUTHOR : IS & BA
**/
```

```
public class OperatorGrpAvg extends JFrame implements WindowListener{

 /** Type of the operator (constant) **/
 final static String OPERATOR_TYPE = "GrpAvg";

 /** Input node number of the operator (constant) **/
 final static int NO_INPUT_NODES = 3;

 /** X coordinate of the operator**/
 private int x;

 /** Y coordinate of the operator**/
 private int y;

 /** name of the operator**/
 private String name;

 /** input node relation**/
 private String relation;

 /** group attribute list**/
 private String groupAtt;

 /** groupping condition **/
 private String aggAtt;

 /** display information for the operator**/
 private boolean display = false;

 /** SQL query of the operator**/
 private String query;

 /** Specify why this frame is called (to create a new one (false)
   * or change the current operator (true)**/
 private boolean whoIsCalling = false;

 /** The index of the operator in the Operators list **/
 private int indexNum;

 /** The index of the operator in the DFQLList**/
 private int current;

 /** name label of the frame**/
 JLabel lblName         = new JLabel();
```

```
/** relation label of the frame**/
JLabel lblRelation      = new JLabel();

/** group attribute list label of the frame**/
JLabel lblGroupAtt  = new JLabel();

/** count attribute list label of the frame**/
JLabel lblAggAtt  = new JLabel();

/** text area to enter the name of the operator default diff#**/
JTextField txtName      = new JTextField();

/** Okay button to confirm the info entered**/
JButton btnOK          = new JButton();

/** check box for display information**/
JCheckBox chkResult     = new JCheckBox();

/** Combo box lists the available operators or table names that can be
  * related to this operator**/
JComboBox cmbRelation1  = new JComboBox();

/** Button cancel to cancel this operation**/
JButton btnCancel       = new JButton();

/** text area to enter the group attribute list of the operator **/
JTextField txtGroupAtt = new JTextField();

JComboBox cmbGroupAtt  = new JComboBox();

/** text area to enter the count attribute list of the operator **/
JTextField txtAggAtt   = new JTextField();

JComboBox cmbAggAtt    = new JComboBox();

/*              CONSTRUCTOR                    */
/**
  * Default constructor
  * @parameters X and Y (integer) coordinates of the operator
**/
public OperatorGrpAvg(int x, int y) {
 try  {
   this.x = x;
   this.y = y;
   jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
 }

/**
  * Constructor with all parameters
  * @parameters @ parameters x, y, caller1, caller2 (integer),
  * name,relation, groupAtt, aggAtt            (String),
  * display                                    (boolean)
**/
public OperatorGrpAvg(int x, int y, int caller1, int caller2, String name,
              String relation, String groupAtt, String aggAtt,
              boolean display){

 this.x           = x;
 this.y           = y;
 this.current     = caller1;
 this.indexNum    = caller2;
 this.whoIsCalling = true;
 this.name        = name;
 this.relation    = relation;
 this.groupAtt    = groupAtt;
 this.aggAtt      = aggAtt;
 this.display     = display;

 txtName.setText          (name);
 txtGroupAtt.setText      (groupAtt);
 txtAggAtt.setText        (aggAtt);

 if (display){
  chkResult.setSelected(true);
 }
 else{
  chkResult.setSelected(false);
 }

 try  {
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}
```

```
/**
  * Design constructor
  * @parameters name
**/
public OperatorGrpAvg(String name) {
  try  {
    this.name = name;
    jbInit();
  }
   catch(Exception e) {
    e.printStackTrace();
  }
}

/*              INITIATE METHOD                    */
/**
  * The initiation
  * creates the frame for user to enter the operator data, and adds
  * buttons, labels, text area and combo boxes.
**/
private void jbInit() throws Exception {
  lblName.setText              ("Name");
  this.setTitle                ("GrpAvg");
  lblRelation.setText          ("Relation");
  lblGroupAtt.setText          ("Group Att.");
  lblAggAtt.setText            ("Agg.Att.");
  btnOK.setText                ("OK");
  btnCancel.setText            ("Cancel");
  chkResult.setText            ("Display");

  lblName.setBounds            (new Rectangle(4, 2, 93, 18));
  lblRelation.setBounds        (new Rectangle(4, 27, 93, 18));
  lblGroupAtt.setBounds        (new Rectangle(4, 51, 93, 18));
  lblAggAtt.setBounds          (new Rectangle(4, 74, 93, 18));
  txtName.setBounds            (new Rectangle(80, 2, 114, 17));
  txtGroupAtt.setBounds        (new Rectangle(80, 52, 114, 17));
  cmbGroupAtt.setBounds        (new Rectangle(80, 52, 114, 17));
  txtAggAtt.setBounds          (new Rectangle(80, 75, 114, 17));
  cmbAggAtt.setBounds          (new Rectangle(80, 75, 114, 17));
  btnOK.setBounds              (new Rectangle(147, 100, 49, 22));
  chkResult.setBounds          (new Rectangle(4, 100, 60, 22));
  cmbRelation1.setBounds       (new Rectangle(80, 28, 114, 17));
  btnCancel.setBounds          (new Rectangle(76, 100, 60, 23));

  if (test.window.isDesign()){
    loadDesign();
```

```
 }

 else{
  loadSelect();
 }

 btnOK.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   btnOK_actionPerformed(e);
  }});

 btnCancel.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   btnCancel_actionPerformed(e);
  }});

 chkResult.setBorder        (BorderFactory.createRaisedBevelBorder());
 btnOK.setBorder            (BorderFactory.createRaisedBevelBorder());
 btnCancel.setBorder        (BorderFactory.createRaisedBevelBorder());

 btnOK.setFont       (new java.awt.Font("Dialog", 0, 11));
 btnCancel.setFont   (new java.awt.Font("Dialog", 0, 11));
 chkResult.setFont   (new java.awt.Font("Dialog", 0, 11));

 chkResult.setBorderPainted         (true);
 cmbRelation1.setBorder             (null);
 this.getContentPane().setLayout    (null);

 this.getContentPane().add(lblName, null);
 this.getContentPane().add(txtName, null);
 this.getContentPane().add(lblRelation, null);
 this.getContentPane().add(cmbRelation1, null);
 this.getContentPane().add(lblGroupAtt, null);

 if(test.window.isDesign()){
  this.getContentPane().add(cmbGroupAtt,null);
  this.getContentPane().add(cmbAggAtt,null);
 }
 else {
  this.getContentPane().add(txtGroupAtt, null);
  this.getContentPane().add(txtAggAtt, null);
 }

 this.getContentPane().add(lblAggAtt, null);
 this.getContentPane().add(chkResult, null);
 this.getContentPane().add(btnOK, null);
```

```
  this.getContentPane().add(btnCancel, null);

  this.setBounds(100,100,210,160);
  this.show(true);
 }

 /*              Set Methods                   */
 /**
  * Sets the relation of the operator
  * @parameter relation (String)
 **/
 public void setRelation (String relation){
  this.relation = relation;
 }

 /**
  * Sets the group aattribute list of the operator
  * @parameter groupAtt (String)
 **/
 public void setGroupAtt (String groupAtt){
  this.groupAtt = groupAtt;
 }

 /**
  * Sets the aggregate attribute list of the operator
  * @parameter aggAtt (String)
 **/
 public void setAggAtt (String aggAtt){
  this.aggAtt = aggAtt;
 }

 /**
  * Sets the name of the operator
  * @parameter name (String)
 **/
 public void setName (String name){
  this.name = name;
 }

 /*              Get Methods                   */
 /**
  * gets the input node number of the operator
 **/
 public int getnumberOfNodes(){
  return this.NO_INPUT_NODES;
 }
```

```
/**
  * gets the type of the operator
**/
public String getOperatorType(){
  return this.OPERATOR_TYPE;
}

/**
  * gets the relation of the operator
**/
public String getRelation (){
  return this.relation;
}

/**
  * gets the group Attribute list of the operator
**/
public String getGroupAtt (){
  return this.groupAtt;
}

/**
  * gets the aggregate attribute of the operator
**/
public String getAggAtt (){
  return this.aggAtt;
}

/**
  * gets the name of the operator
**/
public String getName (){
  return this.name;
}

/*              WINDOW ACTIVATIONS                  */
/**
  * The window listener methods
  * @param event (WindowEvent)
**/
public void windowClosing        (WindowEvent event){System.exit(0);}
public void windowActivated      (WindowEvent event){}
public void windowClosed         (WindowEvent event){}
public void windowDeactivated    (WindowEvent event){}
public void windowDeiconified    (WindowEvent event){}
public void windowIconified        (WindowEvent event){}
public void windowOpened           (WindowEvent event){}

/**
 * button okay action performed
 * Takes the information entered for the operator. Creates a new operator and
 * calls add or change method of the draw class. And sends new operator
 * information there to be added to the list.
 * @param e (ActionEvent)
**/
void btnOK_actionPerformed(ActionEvent e) {
 if (chkResult.isSelected()){
   display = true;
 }
 else{
   display = false;
 }
 setName(txtName.getText());
 setRelation((String)cmbRelation1.getSelectedItem());

 if (test.window.isDesign()){
  setGroupAtt((String)cmbGroupAtt.getSelectedItem());
  setAggAtt ((String)cmbAggAtt.getSelectedItem());
 }
 else {
  setGroupAtt(txtGroupAtt.getText());
  if (txtAggAtt.getText().equals("")){
   test.window.message.setMessage
                          ("You did not enter the aggregate attribute");
   test.window.message.show(true);
  }
  setAggAtt(txtAggAtt.getText());
 }

 String [] input = {getRelation(),getGroupAtt(),getAggAtt(),"none"};
 buildQuery();

 if (test.window.isDesign()){
  test.window.designQueries.addElement(query);
 }
 else {
  if (whoIsCalling){
   test.window.cnvDFQL.change
                   (current,indexNum,x,y,getName(),getnumberOfNodes(),
                    input,display, getOperatorType(),query);
  }
  else{
   test.window.cnvDFQL.add
                    (x,y,getName(),getnumberOfNodes(),input,display,
                     getOperatorType(),query);
  }
 }
 this.dispose();
}

/**
 * Loads the relations with the tables of the selected database and previously
 * created operators.
**/
void loadSelect() {
 Vector deneme = test.window.currentDatabase.getTableNames();

 if(!whoIsCalling){
  txtName.setText("GrpAvg"+test.window.Counter[11]++);
 }

 for (int index = 0 ; index < deneme.size(); index++) {
   cmbRelation1.addItem((String) deneme.elementAt(index));
 }

 for (int xi = 0; xi <test.window.cnvDFQL.numOfOperators;xi++){
  cmbRelation1.addItem(test.window.cnvDFQL.operators[xi].name);
 }
 cmbRelation1.removeItem(name);

 if (relation== null)
   cmbRelation1.setSelectedIndex(0);
 else
   cmbRelation1.setSelectedItem(relation);
}

/**
 * button cancel action performed
 * terminates this frame without creating an operator. decrements the counter
 * @param e (ActionEvent)
**/
void btnCancel_actionPerformed(ActionEvent e) {
  test.window.Counter[11]--;
  this.dispose();
}
```

```
/**
 * This portion of the code produces the SQL query for this operator according
 * to the information entered by user.
**/
public void buildQuery(){
  String checkRelation = isOtherOperator (relation);
  query = "select distinct " + groupAtt;

  if (!aggAtt.equals(""))
    query = query + ", avg ("+aggAtt+") as avg_" + aggAtt;

  if (checkRelation == null) {
     query = query + " from " + relation;
  }
  else {
     query = query + " from " + "(" +checkRelation + ")";
  }

  query = query + " group by " + groupAtt;
}

/**
 * This portion of the code looks for the type of the relations, if they are
 * tables it returns null, otherwise if is another operator, it returns the
 * query of the operator which the current operator related to.
**/
public String isOtherOperator (String type) {
  String result = null;
  for (int index = 0; index < test.window.DFQLQuery.size(); index++) {
    Operator temp = (Operator) test.window.DFQLQuery.elementAt(index);
    if (type.equalsIgnoreCase(temp.name)) {
      result = temp.query;
      break;
    }
  }
  return result;
}

/**
 * This function is used to fill the combo boxes with the selection alternatives
 * if this operator is created as a part of a user defined operator then the
 * relation and condition combo boxes are filled with user defined operator
 * input items.
**/
private void loadDesign(){
  txtName.setText(name);
```

```
    for (int index = 0 ; index < test.window.designRelation.size(); index++) {
     cmbRelation1.addItem((String) test.window.designRelation.elementAt
                                    (index));
    }

    for (int opIndex = 0 ; opIndex < test.window.designOps.size(); opIndex++) {
     if (!name.equalsIgnoreCase((String) test.window.designOps.elementAt
                                        (opIndex)))
       cmbRelation1.addItem((String) test.window.designOps.elementAt
                                      (opIndex));
    }

    for (int attIndex = 0; attIndex < test.window.designAttribute.size();
         attIndex++){
     cmbGroupAtt.addItem((String) test.window.designAttribute.elementAt
                                   (attIndex));
     cmbAggAtt.addItem((String) test.window.designAttribute.elementAt
                                  (attIndex));
    }
  }
}
```

18. OperatorGrpMax.Java

```
package ThesisGUI;

import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;

/**
 * This frame is used to allow the user to enter the operator information
 * for group maximum operator. It also allwos user to change the information
 * previously entered.
 * AUTHOR : IS & BA
**/
public class OperatorGrpMax extends JFrame implements WindowListener{

 /** Type of the operator (constant) **/
 final static String OPERATOR_TYPE = "GrpMax";
```

```
/** Input node number of the operator (constant) **/
final static int NO_INPUT_NODES = 3;

/** X coordinate of the operator**/
private int x;

/** Y coordinate of the operator**/
private int y;

/** name of the operator**/
private String name;

/** input node relation**/
private String relation;

/** group attribute list**/
private String groupAtt;

/** groupping condition **/
private String aggAtt;

/** display information for the operator**/
private boolean display = false;

/** SQL query of the operator**/
private String query;

/** Specify why this frame is called (to create a new one (false)
  * or change the current operator (true)**/
private boolean whoIsCalling = false;

/** The index of the operator in the Operators list **/
private int indexNum;

/** The index of the operator in the DFQLList**/
private int current;

/** name label of the frame**/
JLabel lblName        = new JLabel();

/** relation label of the frame**/
JLabel lblRelation    = new JLabel();

/** group attribute list label of the frame**/
JLabel lblGroupAtt  = new JLabel();
```

```
/** count attribute list label of the frame**/
JLabel lblAggAtt  = new JLabel();

/** text area to enter the name of the operator default diff#**/
JTextField txtName      = new JTextField();

/** Okay button to confirm the info entered**/
JButton btnOK           = new JButton();

/** check box for display information**/
JCheckBox chkResult     = new JCheckBox();

/** Combo box lists the available operators or table names that can be
  * related to this operator**/
JComboBox cmbRelation1  = new JComboBox();

/** Button cancel to cancel this operation**/
JButton btnCancel       = new JButton();

/** text area to enter the group attribute list of the operator **/
JTextField txtGroupAtt  = new JTextField();

JComboBox cmbGroupAtt   = new JComboBox();

/** text area to enter the count attribute list of the operator **/
JTextField txtAggAtt    = new JTextField();

JComboBox cmbAggAtt     = new JComboBox();

/*             CONSTRUCTOR                    */
/**
  * Default constructor
  * @parameters X and Y (integer) coordinates of the operator
**/
public OperatorGrpMax(int x, int y) {
 try {
  this.x = x;
  this.y = y;
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}
```

```
/**
 * Constructor with all parameters
 * @parameters @ parameters x, y, caller1, caller2 (integer),
 * name,relation, groupAtt, aggAtt              (String),
 * display                                     (boolean)
**/
public OperatorGrpMax(int x, int y, int caller1, int caller2, String name,
            String relation, String groupAtt, String aggAtt,
            boolean display){
 this.x             = x;
 this.y             = y;
 this.current       = caller1;
 this.indexNum      = caller2;
 this.whoIsCalling  = true;
 this.name          = name;
 this.relation      = relation;
 this.groupAtt      = groupAtt;
 this.aggAtt        = aggAtt;
 this.display       = display;

 txtName.setText           (name);
 txtGroupAtt.setText       (groupAtt);
 txtAggAtt.setText         (aggAtt);

 if (display){
  chkResult.setSelected(true);
 }
 else{
  chkResult.setSelected(false);
 }

 try {
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}

/**
 * Design constructor
 * @parameters name
**/
public OperatorGrpMax(String name) {
 try {
  this.name = name;
```

```
   jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
}

/*              INITIATE METHOD                    */
/**
 * The initiation
 * creates the frame for user to enter the operator data, and adds
 * buttons, labels, text area and combo boxes.
**/
private void jbInit() throws Exception {
 lblName.setText            ("Name");
 this.setTitle              ("GrpMax");
 lblRelation.setText        ("Relation");
 lblGroupAtt.setText        ("Group Att.");
 lblAggAtt.setText          ("Agg.Att.");
 btnOK.setText              ("OK");
 chkResult.setText          ("Display");
 btnCancel.setText          ("Cancel");

 lblName.setBounds          (new Rectangle(4, 2, 93, 18));
 lblRelation.setBounds      (new Rectangle(4, 27, 93, 18));
 lblGroupAtt.setBounds      (new Rectangle(4, 50, 93, 18));
 lblAggAtt.setBounds        (new Rectangle(4, 73, 93, 18));
 txtName.setBounds          (new Rectangle(80, 2, 114, 17));
 txtGroupAtt.setBounds      (new Rectangle(80, 51, 114, 17));
 cmbGroupAtt.setBounds      (new Rectangle(80, 51, 114, 17));
 txtAggAtt.setBounds        (new Rectangle(80, 74, 114, 17));
 cmbAggAtt.setBounds        (new Rectangle(80, 74, 114, 17));
 btnOK.setBounds            (new Rectangle(143, 99, 52, 22));
 chkResult.setBounds        (new Rectangle(3, 99, 62, 22));
 btnCancel.setBounds        (new Rectangle(74, 99, 60, 23));
 cmbRelation1.setBounds     (new Rectangle(80, 28, 114, 17));

 if (test.window.isDesign()){
  loadDesign();
 }
 else {
  loadSelect();
 }

 btnOK.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
    btnOK_actionPerformed(e);
   }});

  btnCancel.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(ActionEvent e) {
    btnCancel_actionPerformed(e);
   }});

  chkResult.setBorderPainted         (true);
  cmbRelation1.setBorder             (null);

  chkResult.setFont   (new java.awt.Font ("Dialog", 0, 11));
  btnOK.setFont       (new java.awt.Font ("Dialog", 0, 11));
  btnCancel.setFont   (new java.awt.Font ("Dialog", 0, 11));

  btnCancel.setBorder         (BorderFactory.createRaisedBevelBorder());
  btnOK.setBorder             (BorderFactory.createRaisedBevelBorder());
  chkResult.setBorder         (BorderFactory.createRaisedBevelBorder());

  this.getContentPane().setLayout(null);
  this.getContentPane().add(lblName, null);
  this.getContentPane().add(txtName, null);
  this.getContentPane().add(lblRelation, null);
  this.getContentPane().add(cmbRelation1, null);
  this.getContentPane().add(lblGroupAtt, null);
  if(test.window.isDesign()){
    this.getContentPane().add(cmbGroupAtt,null);
    this.getContentPane().add(cmbAggAtt, null);
  }
  else {
    this.getContentPane().add(txtGroupAtt, null);
    this.getContentPane().add(txtAggAtt, null);
  }
  this.getContentPane().add(lblAggAtt, null);
  this.getContentPane().add(btnOK, null);
  this.getContentPane().add(chkResult, null);
  this.getContentPane().add(btnCancel, null);
  this.setBounds(100,100,210,160);
  this.show(true);
}
```

```
/*           Set Methods              */
/**
 * Sets the relation of the operator
 * @parameter relation (String)
**/
public void setRelation (String relation){
 this.relation = relation;
}

/**
 * Sets the group aattribute list of the operator
 * @parameter groupAtt (String)
**/
public void setGroupAtt (String groupAtt){
 this.groupAtt = groupAtt;
}

/**
 * Sets the aggregate attribute list of the operator
 * @parameter aggAtt (String)
**/
public void setAggAtt (String aggAtt){
 this.aggAtt = aggAtt;
}

/**
 * Sets the name of the operator
 * @parameter name (String)
**/
public void setName (String name){
 this.name = name;
}

/*           Get Methods              */
/**
 * gets the input node number of the operator
**/
public int getnumberOfNodes(){
 return this.NO_INPUT_NODES;
}

/**
 * gets the type of the operator
**/
public String getOperatorType(){
 return this.OPERATOR_TYPE;}
```

```
/**
  * gets the relation of the operator
**/
public String getRelation (){
  return this.relation;
}

/**
  * gets the group Attribute list of the operator
**/
public String getGroupAtt (){
  return this.groupAtt;
}

/**
  * gets the aggregate attribute of the operator
**/
public String getAggAtt (){
  return this.aggAtt;
}

/**
  * gets the name of the operator
**/
public String getName (){
  return this.name;
}

/*              WINDOW ACTIVATIONS                    */
/**
  * The window listener methods
  * @param event (WindowEvent)
**/
public void windowClosing          (WindowEvent event){System.exit(0);}
public void windowActivated        (WindowEvent event){}
public void windowClosed           (WindowEvent event){}
public void windowDeactivated      (WindowEvent event){}
public void windowDeiconified      (WindowEvent event){}
public void windowIconified        (WindowEvent event){}
public void windowOpened           (WindowEvent event){}
```

```
/**
  * button okay action performed
  * Takes the information entered for the operator. Creates a new operator and
  * calls add or change method of the draw class. And sends new operator
  * information there to be added to the list.
  * @param e (ActionEvent)
**/
void btnOK_actionPerformed(ActionEvent e) {
 if(chkResult.isSelected()){
    display = true;
 }
 else{
  display = false;
 }
 setName(txtName.getText());
 setRelation((String)cmbRelation1.getSelectedItem());

 if (test.window.isDesign()){
   setGroupAtt((String)cmbGroupAtt.getSelectedItem());
   setAggAtt((String)cmbAggAtt.getSelectedItem());
 }
 else {
   setGroupAtt(txtGroupAtt.getText());
   if (txtAggAtt.getText().equals("")){
     test.window.message.setMessage
                              ("You did not enter the aggregate attribute");
     test.window.message.show(true);
   }
   setAggAtt(txtAggAtt.getText());
 }
 String [] input = {getRelation(),getGroupAtt(),getAggAtt(),"none"};
 buildQuery();
 if (test.window.isDesign()){
   test.window.designQueries.addElement(query);
 }
 else {
  if (whoIsCalling){
    test.window.cnvDFQL.change
                         (current,indexNum,x,y,getName(),getnumberOfNodes(),
                          input,display, getOperatorType(),query);
  }
  else{
    test.window.cnvDFQL.add
                        (x,y,getName(),getnumberOfNodes(),input,display,
                         getOperatorType(), query);
  }
```

```
    }
    this.dispose();
  }

  /**
   * Loads the relations with the tables of the selected database and previously
   * created operators.
  **/
  void loadSelect() {
    Vector deneme = test.window.currentDatabase.getTableNames();

    if(!whoIsCalling){
      txtName.setText("GrpMax"+test.window.Counter[9]++);
    }

    for (int index = 0 ; index < deneme.size(); index++) {
      cmbRelation1.addItem((String) deneme.elementAt(index));
    }

    for (int xi = 0; xi <test.window.cnvDFQL.numOfOperators;xi++){
      cmbRelation1.addItem(test.window.cnvDFQL.operators[xi].name);
    }
    cmbRelation1.removeItem(name);

    if (relation== null)
      cmbRelation1.setSelectedIndex(0);
    else
      cmbRelation1.setSelectedItem(relation);
  }

  /**
   * button cancel action performed
   * terminates this frame without creating an operator. decrements the counter
   * @param e (ActionEvent)
  **/
  void btnCancel_actionPerformed(ActionEvent e) {
     test.window.Counter[9]--;
     this.dispose();
  }

  /**
   * This portion of the code produces the SQL query for this operator according
   * to the information entered by user.
  **/
  public void buildQuery(){
    String checkRelation = isOtherOperator (relation);
    query = "select distinct " + groupAtt;

    if (!aggAtt.equals(""))
      query = query + ", max ("+aggAtt+") as max_" + aggAtt;

    if (checkRelation == null) {
       query = query + " from " + relation;
    }
    else {
       query = query + " from " + "(" +checkRelation + ")";
    }

    query = query + " group by " + groupAtt;
  }

  /**
   * This portion of the code looks for the type of the relations, if they are
   * tables it returns null, otherwise if is another operator, it returns the
   * query of the operator which the current operator related to.
  **/
  public String isOtherOperator (String type) {
    String result = null;
    for (int index = 0; index < test.window.DFQLQuery.size(); index++) {
      Operator temp = (Operator) test.window.DFQLQuery.elementAt(index);
      if (type.equalsIgnoreCase(temp.name)) {
        result = temp.query;
        break;
      }
    }
    return result;
  }

  /**
   * This function is used to fill the combo boxes with the selection alternatives
   * if this operator is created as a part of a user defined operator then the
   * relation and condition combo boxes are filled with user defined operator
   * input items.
  **/
  private void loadDesign(){
    txtName.setText(name);
    for (int index = 0 ; index < test.window.designRelation.size(); index++) {
      cmbRelation1.addItem((String) test.window.designRelation.elementAt
                                   (index));
    }

    for (int opIndex = 0 ; opIndex < test.window.designOps.size(); opIndex++) {
      if (!name.equalsIgnoreCase((String) test.window.designOps.elementAt
                                          (opIndex)))
        cmbRelation1.addItem((String) test.window.designOps.elementAt
                                       (opIndex));
    }

    for (int attIndex = 0; attIndex < test.window.designAttribute.size();
         attIndex++){
      cmbGroupAtt.addItem((String) test.window.designAttribute.elementAt
                                    (attIndex));

      cmbAggAtt.addItem((String) test.window.designAttribute.elementAt
                                  (attIndex));
    }
  }
}
```

19. OperatorGrpMin.Java

```
package ThesisGUI;

import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;

/**
 * This frame is used to allow the user to enter the operator information
 * for group minimum operator. It also allwos user to change the information
 * previously entered.
 * AUTHOR : IS & BA
**/
public class OperatorGrpMin extends JFrame implements WindowListener{

  /** Type of the operator (constant) **/
  final static String OPERATOR_TYPE = "GrpMin";

  /** Input node number of the operator (constant) **/
  final static int NO_INPUT_NODES = 3;

  /** X coordinate of the operator**/
  private int x;
```

```
/** Y coordinate of the operator**/
private int y;

/** name of the operator**/
private String name;

/** input node relation**/
private String relation;

/** group attribute list**/
private String groupAtt;

/** groupping condition **/
private String aggAtt;

/** display information for the operator**/
private boolean display = false;

/** SQL query of the operator**/
private String query;

/** Specify why this frame is called (to create a new one (false)
  * or change the current operator (true)**/
private boolean whoIsCalling = false;

/** The index of the operator in the Operators list **/
private int indexNum;

/** The index of the operator in the DFQLList**/
private int current;

/** name label of the frame**/
JLabel lblName          = new JLabel();

/** relation label of the frame**/
JLabel lblRelation      = new JLabel();

/** group attribute list label of the frame**/
JLabel lblGroupAtt  = new JLabel();

/** count attribute list label of the frame**/
JLabel lblAggAtt   = new JLabel();

/** text area to enter the name of the operator default diff#**/
JTextField txtName      = new JTextField();
```

```
/** Okay button to confirm the info entered**/
JButton btnOK          = new JButton();

/** check box for display information**/
JCheckBox chkResult    = new JCheckBox();

/** Combo box lists the available operators or table names that can be
 * related to this operator**/
JComboBox cmbRelation1  = new JComboBox();

/** Button cancel to cancel this operation**/
JButton btnCancel       = new JButton();

/** text area to enter the group attribute list of the operator **/
JTextField txtGroupAtt = new JTextField();

JComboBox cmbGroupAtt   = new JComboBox();

/** text area to enter the count attribute list of the operator **/
JTextField txtAggAtt = new JTextField();

JComboBox cmbAggAtt     = new JComboBox();

/*           CONSTRUCTOR                */
/**
 * Default constructor
 * @parameters X and Y (integer) coordinates of the operator
**/
public OperatorGrpMin(int x, int y) {
 try {
  this.x = x;
  this.y = y;
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}
```

```
  /**
   * Constructor with all parameters
   * @parameters @ parameters x, y, caller1, caller2 (integer),
   * name,relation, groupAtt, aggAtt              (String),
   * display                                      (boolean)
  **/
  public OperatorGrpMin(int x, int y, int caller1, int caller2, String name,
                String relation, String groupAtt, String aggAtt,
                boolean display){
   this.x             = x;
   this.y             = y;
   this.current       = caller1;
   this.indexNum      = caller2;
   this.whoIsCalling  = true;
   this.name          = name;
   this.relation      = relation;
   this.groupAtt      = groupAtt;
   this.aggAtt        = aggAtt;
   this.display       = display;

   txtName.setText          (name);
   txtGroupAtt.setText      (groupAtt);
   txtAggAtt.setText        (aggAtt);

   if (display){
    chkResult.setSelected(true);
   }
   else{
    chkResult.setSelected(false);
   }

   try {
    jbInit();
   }
   catch(Exception e) {
    e.printStackTrace();
   }
 }

 /**
  * Design constructor
  * @parameters name
 **/
 public OperatorGrpMin(String name) {
  try {
   this.name = name;
   jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
 }

/*              INITIATE METHOD                    */
/**
 * The initiation
 * creates the frame for user to enter the operator data, and adds
 * buttons, labels, text area and combo boxes.
**/
private void jbInit() throws Exception {
 lblName.setText           ("Name");
 this.setTitle             ("GrpMin");
 lblRelation.setText       ("Relation");
 lblGroupAtt.setText       ("Group Att.");
 lblAggAtt.setText         ("Agg.Att.");
 btnOK.setText             ("OK");
 btnCancel.setText         ("Cancel");
 chkResult.setText         ("Display");

 lblName.setBounds         (new Rectangle(4, 2, 93, 18));
 lblRelation.setBounds     (new Rectangle(4, 27, 93, 18));
 lblGroupAtt.setBounds     (new Rectangle(4, 50, 93, 18));
 lblAggAtt.setBounds       (new Rectangle(4, 73, 93, 18));
 txtName.setBounds         (new Rectangle(80, 2, 114, 17));
 txtGroupAtt.setBounds     (new Rectangle(80, 51, 114, 17));
 txtAggAtt.setBounds       (new Rectangle(80, 74, 114, 17));
 cmbGroupAtt.setBounds     (new Rectangle(80, 51, 114, 17));
 cmbAggAtt.setBounds       (new Rectangle(80, 74, 114, 17));
 btnOK.setBounds           (new Rectangle(147, 100, 49, 22));
 chkResult.setBounds       (new Rectangle(3, 100, 59, 22));
 cmbRelation1.setBounds    (new Rectangle(80, 28, 114, 17));
 btnCancel.setBounds       (new Rectangle(125, 140, 60, 23));
 btnCancel.setBounds       (new Rectangle(68, 100, 73, 23));

 if (test.window.isDesign()){
  loadDesign();
 }
 else {
  loadSelect();
 }

 btnOK.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(ActionEvent e) {
    btnOK_actionPerformed(e);
   }});

  btnCancel.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(ActionEvent e) {
    btnCancel_actionPerformed(e);
   }});

  btnOK.setFont       (new java.awt.Font("Dialog", 0, 11));
  chkResult.setFont   (new java.awt.Font("Dialog", 0, 11));
  btnCancel.setFont   (new java.awt.Font("Dialog", 0, 11));

  btnCancel.setBorder        (BorderFactory.createRaisedBevelBorder());
  btnOK.setBorder            (BorderFactory.createRaisedBevelBorder());
  chkResult.setBorder        (BorderFactory.createRaisedBevelBorder());

  chkResult.setBorderPainted        (true);
  cmbRelation1.setBorder            (null);
  this.getContentPane().setLayout   (null);

  this.getContentPane().add(lblName, null);
  this.getContentPane().add(txtName, null);
  this.getContentPane().add(lblRelation, null);
  this.getContentPane().add(cmbRelation1, null);
  this.getContentPane().add(lblGroupAtt, null);
  this.getContentPane().add(lblAggAtt, null);
  if(test.window.isDesign()){
    this.getContentPane().add(cmbGroupAtt,null);
    this.getContentPane().add(cmbAggAtt, null);
  }
  else {
    this.getContentPane().add(txtGroupAtt, null);
    this.getContentPane().add(txtAggAtt, null);
  }
  this.getContentPane().add(chkResult, null);
  this.getContentPane().add(btnOK, null);
  this.getContentPane().add(btnCancel, null);

  this.setBounds(100,100,210,160);
  this.show(true);
 }
```

```
/*              Set Methods                    */
/**
  * Sets the relation of the operator
  * @parameter relation (String)
**/
public void setRelation (String relation){
  this.relation = relation;
}

/**
  * Sets the group aattribute list of the operator
  * @parameter groupAtt (String)
**/
public void setGroupAtt (String groupAtt){
  this.groupAtt = groupAtt;
}

/**
  * Sets the aggregate attribute list of the operator
  * @parameter aggAtt (String)
**/
public void setAggAtt (String aggAtt){
  this.aggAtt = aggAtt;
}

/**
  * Sets the name of the operator
  * @parameter name (String)
**/
public void setName (String name){
  this.name = name;
}

/*              Get Methods                    */
/**
  * gets the input node number of the operator
**/
public int getnumberOfNodes(){
  return this.NO_INPUT_NODES;
}

/**
  * gets the type of the operator
**/
public String getOperatorType(){
  return this.OPERATOR_TYPE; }
```

```
/**
  * gets the relation of the operator
**/
public String getRelation (){
  return this.relation;
}

/**
  * gets the group Attribute list of the operator
**/
public String getGroupAtt (){
  return this.groupAtt;
}

/**
  * gets the aggregate attribute of the operator
**/
public String getAggAtt (){
  return this.aggAtt;
}

/**
  * gets the name of the operator
**/
public String getName (){
  return this.name;
}

/*          WINDOW ACTIVATIONS                    */
/**
  * The window listener methods
  * @param event (WindowEvent)
**/
public void windowClosing        (WindowEvent event){System.exit(0);}
public void windowActivated      (WindowEvent event){}
public void windowClosed         (WindowEvent event){}
public void windowDeactivated    (WindowEvent event){}
public void windowDeiconified    (WindowEvent event){}
public void windowIconified      (WindowEvent event){}
public void windowOpened         (WindowEvent event){}
```

```
/**
 * button okay action performed
 * Takes the information entered for the operator. Creates a new operator and
 * calls add or change method of the draw class. And sends new operator
 * information there to be added to the list.
 * @param e (ActionEvent)
**/
void btnOK_actionPerformed(ActionEvent e) {
 if(chkResult.isSelected()){
    display = true;
 }
 else{
   display = false;
 }
 setName(txtName.getText());
 setRelation((String)cmbRelation1.getSelectedItem());

 if (test.window.isDesign()){
   setGroupAtt((String)cmbGroupAtt.getSelectedItem());
   setAggAtt((String)cmbAggAtt.getSelectedItem());
 }
 else {
   setGroupAtt(txtGroupAtt.getText());
   if (txtAggAtt.getText().equals("")){
     test.window.message.setMessage
                            ("You did not enter the aggregate attribute");
     test.window.message.show(true);
   }
   setAggAtt(txtAggAtt.getText());
 }

 String [] input = {getRelation(),getGroupAtt(),getAggAtt(),"none"};
 buildQuery();

 if (test.window.isDesign()){
   test.window.designQueries.addElement(query);
 }
 else {
   if (whoIsCalling){
     test.window.cnvDFQL.change
                    (current,indexNum,x,y,getName(),getnumberOfNodes(),
                     input,display, getOperatorType(), query);
   }
   else {
     test.window.cnvDFQL.add
                    (x,y,getName(),getnumberOfNodes(),input,display,
                    getOperatorType(), query);
     }
   }
   this.dispose();
  }

 /**
  * Loads the relations with the tables of the selected database and previously
  * created operators.
 **/
 void loadSelect() {
  Vector deneme = test.window.currentDatabase.getTableNames();

  if(!whoIsCalling){
   txtName.setText("GrpMin"+test.window.Counter[10]++);
  }

  for (int index = 0 ; index < deneme.size(); index++) {
    cmbRelation1.addItem((String) deneme.elementAt(index));
  }

  for (int xi = 0; xi <test.window.cnvDFQL.numOfOperators;xi++){
   cmbRelation1.addItem(test.window.cnvDFQL.operators[xi].name);
  }
  cmbRelation1.removeItem(name);

  if (relation== null)
    cmbRelation1.setSelectedIndex(0);
  else
    cmbRelation1.setSelectedItem(relation);
 }

 /**
  * button cancel action performed
  * terminates this frame without creating an operator. decrements the counter
  * @param e (ActionEvent)
 **/
 void btnCancel_actionPerformed(ActionEvent e) {
   test.window.Counter[10]--;
   this.dispose();
 }
```

```
/**
 * This portion of the code produces the SQL query for this operator according
 * to the information entered by user.
**/
public void buildQuery(){
  String checkRelation = isOtherOperator (relation);
  query = "select distinct " + groupAtt;

  if (!aggAtt.equals(""))
   query = query + ", min ("+aggAtt+") as min_" + aggAtt;

  if (checkRelation == null) {
     query = query + " from " + relation;
  }
  else {
     query = query + " from " + "(" +checkRelation + ")";
  }

  query = query + " group by " + groupAtt;
}

/**
 * This portion of the code looks for the type of the relations, if they are
 * tables it returns null, otherwise if is another operator, it returns the
 * query of the operator which the current operator related to.
**/
public String isOtherOperator (String type) {
  String result = null;
  for (int index = 0; index < test.window.DFQLQuery.size(); index++) {
    Operator temp = (Operator) test.window.DFQLQuery.elementAt(index);
    if (type.equalsIgnoreCase(temp.name)) {
      result = temp.query;
      break;
    }
  }
  return result;
}

/**
 * This function is used to fill the combo boxes with the selection alternatives
 * if this operator is created as a part of a user defined operator then the
 * relation and condition combo boxes are filled with user defined operator
 * input items.
**/
private void loadDesign(){
  txtName.setText(name);
```

```
    for (int index = 0 ; index < test.window.designRelation.size(); index++) {
     cmbRelation1.addItem((String) test.window.designRelation.elementAt
                                  (index));
    }

    for (int opIndex = 0 ; opIndex < test.window.designOps.size(); opIndex++) {
     if (!name.equalsIgnoreCase((String) test.window.designOps.elementAt
                                      (opIndex)))
       cmbRelation1.addItem((String) test.window.designOps.elementAt
                                    (opIndex));
    }

    for (int attIndex = 0; attIndex < test.window.designAttribute.size();
         attIndex++){
     cmbGroupAtt.addItem((String) test.window.designAttribute.elementAt
                                  (attIndex));
     cmbAggAtt.addItem((String) test.window.designAttribute.elementAt
                                (attIndex));
    }
  }
}
```

20. OperatorGrpNSat.Java

```
package ThesisGUI;

import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;

/**
 * This frame is used to allow the user to enter the operator information
 * for group number satisfy operator. It also allwos user to change
 * the information previously entered.
 * AUTHOR : IS & BA
**/
public class OperatorGrpNSat extends JFrame implements WindowListener{

 /** Type of the operator (constant) **/
 final static String OPERATOR_TYPE = "GrpNSat";
```

```
/** Input node number of the operator (constant) **/
final static int NO_INPUT_NODES = 4;

/** X coordinate of the operator**/
private int x;

/** Y coordinate of the operator**/
private int y;

/** name of the operator**/
private String name;

/** input node relation**/
private String relation;

/** condition number **/
private String number;

/** group attribute list**/
private String groupAtt;

/** groupping condition **/
private String condition;

/** display information for the operator**/
private boolean display = false;

/** SQL query of the operator**/
private String query;

/** Specify why this frame is called (to create a new one (false)
  * or change the current operator (true)**/
private boolean whoIsCalling = false;

/** The index of the operator in the Operators list **/
private int indexNum;

/** The index of the operator in the DFQLList**/
private int current;

/** name label of the frame**/
JLabel lblName         = new JLabel();

/** relation label of the frame**/
JLabel lblRelation     = new JLabel();
```

```
 /** group attribute list label of the frame**/
 JLabel lblGroupAtt  = new JLabel();

 /** count attribute list label of the frame**/
 JLabel lblCondition   = new JLabel();

 /** condition number of the frame**/
 JLabel lblNumber   = new JLabel();

 /** text area to enter the name of the operator default diff#**/
 JTextField txtName       = new JTextField();

 /** Okay button to confirm the info entered**/
 JButton btnOK           = new JButton();

 /** check box for display information**/
 JCheckBox chkResult     = new JCheckBox();

 /** Combo box lists the available operators or table names that can be
   * related to this operator**/
 JComboBox cmbRelation1  = new JComboBox();

 /** Button cancel to cancel this operation**/
 JButton btnCancel       = new JButton();

 /** text area to enter the group attribute list of the operator **/
 JTextField txtGroupAtt = new JTextField();

 /** text area to enter the count attribute list of the operator **/
 JTextField txtCondition = new JTextField();

 /** text area to enter the condition number of the operator **/
 JTextField txtNumber = new JTextField();

 /*            CONSTRUCTOR                     */
 /**
   * Default constructor
   * @parameters X and Y (integer) coordinates of the operator
 **/
 public OperatorGrpNSat(int x, int y) {
  try  {
   this.x = x;
   this.y = y;
   jbInit();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
 }

 /**
  * Constructor with all parameters
  * @parameters @ parameters x, y, caller1, caller2 (integer),
  * name,relation, groupAtt, condition, number      (String),
  * display                                    (boolean)
**/
public OperatorGrpNSat(int x, int y, int caller1, int caller2, String name,
                String relation, String groupAtt, String condition,
                String number, boolean display){
  this.x              = x;
  this.y              = y;
  this.current        = caller1;
  this.indexNum       = caller2;
  this.whoIsCalling   = true;
  this.name           = name;
  this.relation       = relation;
  this.groupAtt       = groupAtt;
  this.condition      = condition;
  this.number         = number;
  this.display        = display;

  txtName.setText           (name);
  txtGroupAtt.setText       (groupAtt);
  txtCondition.setText      (condition);
  txtNumber.setText         (number);

  if (display){
   chkResult.setSelected(true);
  }
  else{
   chkResult.setSelected(false);
  }

  try  {
   jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
 }
```

```
/*              INITIATE METHOD                    */
/**
 * The initiation
 * creates the frame for user to enter the operator data, and adds
 * buttons, labels, text area and combo boxes.
**/
private void jbInit() throws Exception {
 lblName.setText            ("Name");
 this.setTitle              ("GrpNSat");
 lblRelation.setText        ("Relation");
 lblGroupAtt.setText        ("Group Att.");
 lblCondition.setText       ("Condition");
 lblNumber.setText          ("Number");
 btnOK.setText              ("OK");
 chkResult.setText          ("Display");
 btnCancel.setText          ("Cancel");

 lblName.setBounds          (new Rectangle(4, 2, 93, 18));
 lblRelation.setBounds      (new Rectangle(3, 24, 93, 18));
 lblGroupAtt.setBounds      (new Rectangle(4, 47, 93, 18));
 lblCondition.setBounds     (new Rectangle(4, 70, 93, 18));
 txtName.setBounds          (new Rectangle(80, 2, 114, 17));
 txtGroupAtt.setBounds      (new Rectangle(80, 48, 114, 17));
 txtCondition.setBounds     (new Rectangle(80, 71, 114, 17));
 btnOK.setBounds            (new Rectangle(142, 117, 51, 22));
 chkResult.setBounds        (new Rectangle(5, 117, 58, 22));
 cmbRelation1.setBounds     (new Rectangle(79, 25, 114, 17));
 lblNumber.setBounds        (new Rectangle(4, 95, 68, 16));
 txtNumber.setBounds        (new Rectangle(80, 95, 114, 17));
 btnCancel.setBounds        (new Rectangle(73, 117, 60, 23));

 if(!whoIsCalling){
  txtName.setText("GrpNSat"+test.window.Counter[8]++);
 }
 loadSelect();

 btnOK.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   btnOK_actionPerformed(e);
  }});

 btnCancel.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   btnCancel_actionPerformed(e);
  }});
 chkResult.setBorderPainted          (true);
 cmbRelation1.setBorder              (null);

 chkResult.setFont   (new java.awt.Font("Dialog", 0, 11));
 btnOK.setFont       (new java.awt.Font("Dialog", 0, 11));
 btnCancel.setFont   (new java.awt.Font("Dialog", 0, 11));

 btnOK.setBorder               (BorderFactory.createRaisedBevelBorder());
 chkResult.setBorder           (BorderFactory.createRaisedBevelBorder());
 btnCancel.setBorder           (BorderFactory.createRaisedBevelBorder());

 this.getContentPane().setLayout    (null);
 this.getContentPane().add(lblName, null);
 this.getContentPane().add(txtName, null);
 this.getContentPane().add(lblRelation, null);
 this.getContentPane().add(cmbRelation1, null);
 this.getContentPane().add(txtGroupAtt, null);
 this.getContentPane().add(lblGroupAtt, null);
 this.getContentPane().add(lblCondition, null);
 this.getContentPane().add(txtCondition, null);
 this.getContentPane().add(lblNumber, null);
 this.getContentPane().add(txtNumber, null);
 this.getContentPane().add(chkResult, null);
 this.getContentPane().add(btnCancel, null);
 this.getContentPane().add(btnOK, null);

 this.setBounds(100,100,210,180);
 this.show(true);
}

/*            Set Methods                */
/**
 * Sets the relation of the operator
 * @parameter relation (String)
**/
public void setRelation (String relation){
 this.relation = relation;
}

/**
 * Sets the group attribute list of the operator
 * @parameter groupAtt (String)
**/
public void setGroupAtt (String groupAtt){
 this.groupAtt = groupAtt;
}
```

```
/**
 * Sets the condition of the operator
 * @parameter condition (String)
**/
public void setCondition (String condition){
 this.condition = condition;
}

/**
 * Sets the condition number of the operator
 * @parameter condition (String)
**/
public void setNumber (String number){
 this.number = number;
}

/**
 * Sets the name of the operator
 * @parameter name (String)
**/
public void setName (String name){
 this.name = name;
}

/*              Get Methods                    */
/**
 * gets the input number of the operator
**/
public int getnumberOfNodes(){
 return this.NO_INPUT_NODES;
}

/**
 * gets the type of the operator
**/
public String getOperatorType(){
 return this.OPERATOR_TYPE;
}

/**
 * gets the relation of the operator
**/
public String getRelation (){
 return this.relation;
}
```

```
/**
 * gets the group Attribute list of the operator
**/
public String getGroupAtt (){
 return this.groupAtt;
}

/**
 * gets the condition of the operator
**/
public String getCondition (){
 return this.condition;
}

/**
 * gets the name of the operator
**/
public String getName (){
 return this.name;
}

/**
 * gets the condition number of the operator
**/
public String getNumber (){
 return this.number;
}

/*          WINDOW ACTIVATIONS                    */
/**
 * The window listener methods
 * @param event (WindowEvent)
**/
public void windowClosing       (WindowEvent event){System.exit(0);}
public void windowActivated     (WindowEvent event){}
public void windowClosed        (WindowEvent event){}
public void windowDeactivated   (WindowEvent event){}
public void windowDeiconified   (WindowEvent event){}
public void windowIconified     (WindowEvent event){}
public void windowOpened        (WindowEvent event){}
```

```
/**
 * button okay action performed
 * Takes the information entered for the operator. Creates a new operator and
 * calls add or change method of the draw class. And sends new operator
 * information there to be added to the list.
 * @param e (ActionEvent)
**/
void btnOK_actionPerformed(ActionEvent e) {
 if(chkResult.isSelected())
 {
    display = true;
 }
 else
 {
   display = false;
 }
 setName(txtName.getText());
 setRelation((String)cmbRelation1.getSelectedItem());
 setGroupAtt(txtGroupAtt.getText());

 if ((txtCondition.getText().equals(""))||(txtNumber.getText().equals(""))){
   test.window.message.setMessage
                    ("You did not enter the condition or count number");
   test.window.message.show(true);
 }

 setCondition(txtCondition.getText());
 setNumber(txtNumber.getText());
 String [] input = {getRelation(),getGroupAtt(),getCondition(),getNumber()};
 buildQuery();
 if (whoIsCalling)
 {
  test.window.cnvDFQL.change
                    (current,indexNum,x,y,getName(),getnumberOfNodes(),
                     input,display, getOperatorType(), query);
 }
 else
 {
  test.window.cnvDFQL.add
                    (x,y,getName(),getnumberOfNodes(),input,display,
                     getOperatorType(), query);
 }
 this.dispose();
}
```

```
/**
  * Loads the relations with the tables of the selected database and previously
  * created operators.
 **/
 void loadSelect() {
  Vector deneme = test.window.currentDatabase.getTableNames();
  for (int index = 0 ; index < deneme.size(); index++) {
    cmbRelation1.addItem((String) deneme.elementAt(index));
  }

  for (int xi = 0; xi <test.window.cnvDFQL.numOfOperators;xi++){
   cmbRelation1.addItem(test.window.cnvDFQL.operators[xi].name);
  }
  cmbRelation1.removeItem(name);

  if (relation== null)
    cmbRelation1.setSelectedIndex(0);
  else
    cmbRelation1.setSelectedItem(relation);
 }

/**
  * button cancel action performed
  * terminates this frame without creating an operator. decrements the counter
  * @param e (ActionEvent)
**/
void btnCancel_actionPerformed(ActionEvent e) {
   test.window.Counter[8]--;
   this.dispose();
}

/**
  * This portion of the code produces the SQL query for this operator according
  * to the information entered by user.
**/
public void buildQuery(){
  String checkRelation = isOtherOperator (relation);
  query = "select distinct " + groupAtt;

  if (checkRelation == null) {
     query = query + " from " + relation;
  }
  else {
     query = query + " from " + "(" +checkRelation + ")";
  }
```

```
      if (!condition.equals("")) {
        query = query + " where " + condition;
      }

      query = query + " group by " + groupAtt;

      if (!number.equals(""))
        query = query + " having count (*) " +  number;
    }

   /**
    * This portion of the code looks for the type of the relations, if they are
    * tables it returns null, otherwise if is another operator, it returns the
    * query of the operator which the current operator related to.
   **/
   public String isOtherOperator (String type) {
     String result = null;
     for (int index = 0; index < test.window.DFQLQuery.size(); index++) {
       Operator temp = (Operator) test.window.DFQLQuery.elementAt(index);
       if (type.equalsIgnoreCase(temp.name)) {
         result = temp.query;
         break;
       }
     }
   return result;
   }
  }
```

21. OperatorIntersect.Java

```
package ThesisGUI;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;

/**
 * This frame is used to allow the user to enter the operator information
 * for Intersect operator. It also allwos user to change the information
 * previously entered.
 * AUTHOR : IS & BA
**/
```

```
public class OperatorIntersect extends JFrame implements WindowListener{
 /** Type of the operator (constant) **/
 final static String OPERATOR_TYPE = "Intersect";

 /** Input node number of the operator (constant) **/
 final static int NO_INPUT_NODES = 3;

 /** X coordinate of the operator**/
 private int x;

 /** Y coordinate of the operator**/
 private int y;

 /** name of the operator**/
 private String name;

 /** first input node relation**/
 private String relation1;

 /** second input node relation**/
 private String relation2;

 /** common attribute name (intersect point )**/
 private String attribute;

 /** display information for the operator**/
 private boolean display = false;

 /** SQL query of the operator**/
 private String query;

 /** Specify why this frame is called (to create a new one (false)
   * or change the current operator (true)**/
 private boolean whoIsCalling = false;

 /** The index of the operator in the Operators list **/
 private int indexNum;

 /** The index of the operator in the DFQLList**/
 private int current;

 /** name label of the frame**/
 JLabel lblName        = new JLabel();

 /** first relation label of the frame**/
 JLabel lblRelation1    = new JLabel();
```

```
/** second relation label of the frame**/
JLabel lblRelation2     = new JLabel();

/** common attribute label of the frame**/
JLabel lblAttribute       = new JLabel();

/** text area to enter the name of the operator default diff#**/
JTextField txtName      = new JTextField();

/** text area to enter the name of the intersect attribute**/
JTextField txtAttribute   = new JTextField();

/** check box for display information**/
JCheckBox chkResult     = new JCheckBox();

/** Combo box lists the available operators or table names that can be
  * related to this operator**/
JComboBox cmbRelation1  = new JComboBox();

/** Combo box lists the available operators or table names that can be
  * related to this operator**/
JComboBox cmbRelation2  = new JComboBox();

/** Combo box lists the available attributes that can be
  * related to this operator**/
JComboBox cmbAttribute     = new JComboBox();

 /** Okay button to confirm the info entered**/
JButton btnOK          = new JButton();

/** Button cancel to cancel this operation**/
JButton btnCancel       = new JButton();

/*              CONSTRUCTOR                    .  */
/**
  * Default constructor
  * @parameters X and Y (integer) coordinates of the operator
**/
public OperatorIntersect(int x, int y) {
  try  {
   this.x = x;
   this.y = y;
   jbInit();
  }
  catch(Exception e) {
```

```
  e.printStackTrace();
 }
}

 /**
  * Constructor with all parameters
  * @parameters @ parameters x, y, caller1, caller2 (integer),
  * name,relation1, relation2, attribute          (String),
  * display                                        (boolean)
**/
public OperatorIntersect(int x, int y, int caller1, int caller2, String name,
                String relation1, String relation2, String attribute, boolean
                display)
{
 this.x              = x;
 this.y              = y;
 this.current        = caller1;
 this.indexNum       = caller2;
 this.whoIsCalling   = true;
 this.name           = name;
 this.relation1      = relation1;
 this.relation2      = relation2;
 this.attribute      = attribute;
 this.display        = display;

 txtName.setText     (name);
 txtAttribute.setText(attribute);
 if (display){
  chkResult.setSelected(true);
 }
 else{
  chkResult.setSelected(false);
 }

 try {
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}
```

```
/**
  * Design constructor
  * @parameters name
**/
public OperatorIntersect(String name) {
 try  {
  this.name = name;
  jbInit();
 }
  catch(Exception e) {
   e.printStackTrace();
 }
}

/*              INITIATE METHOD                    */
/**
 * The initiation
 * creates the frame for user to enter the operator data, and adds
 * buttons, labels, text area and combo boxes.
**/
private void jbInit() throws Exception {
 lblName.setText            ("Name");
 this.setTitle              ("Intersect");
 lblRelation1.setText       ("Relation1");
 lblRelation2.setText       ("Relation2");
 lblAttribute.setText       ("Attribute");
 btnOK.setText              ("OK");
 btnCancel.setText          ("Cancel");
 chkResult.setText          ("Display");

 lblName.setBounds          (new Rectangle(4, 2, 93, 18));
 lblRelation1.setBounds     (new Rectangle(4, 27, 93, 18));
 lblRelation2.setBounds     (new Rectangle(4, 50, 93, 18));
 lblAttribute.setBounds     (new Rectangle(4, 73, 93, 18));
 txtName.setBounds          (new Rectangle(80, 2 , 114, 17));
 txtAttribute.setBounds     (new Rectangle(80, 73, 114, 17));
 btnOK.setBounds            (new Rectangle(142, 100, 54, 22));
 btnCancel.setBounds        (new Rectangle(74, 100, 60, 23));
 chkResult.setBounds        (new Rectangle(4, 100, 61, 22));
 cmbRelation1.setBounds     (new Rectangle(80, 28, 114, 17));
 cmbRelation2.setBounds     (new Rectangle(79, 51, 114, 17));
 cmbAttribute.setBounds     (new Rectangle(80, 73, 114, 17));

 if (test.window.isDesign()){
  loadDesign();
 }
else {
 loadSelect();
}

btnOK.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnOK_actionPerformed(e);
 }});

btnCancel.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnCancel_actionPerformed(e);
 }});

cmbRelation1.setBorder          (null);
cmbRelation2.setBorder          (null);
chkResult.setBorderPainted      (true);

chkResult.setFont   (new java.awt.Font("Dialog", 0, 11));
btnCancel.setFont   (new java.awt.Font("Dialog", 0, 11));
btnOK.setFont       (new java.awt.Font("Dialog", 0, 11));

btnOK.setBorder            (BorderFactory.createRaisedBevelBorder());
chkResult.setBorder        (BorderFactory.createRaisedBevelBorder());
btnCancel.setBorder        (BorderFactory.createRaisedBevelBorder());

this.getContentPane().setLayout(null);
this.getContentPane().add(lblName, null);
this.getContentPane().add(txtName, null);
this.getContentPane().add(lblRelation1, null);
this.getContentPane().add(cmbRelation1, null);
this.getContentPane().add(cmbRelation2, null);
this.getContentPane().add(lblRelation2, null);
this.getContentPane().add(chkResult, null);
this.getContentPane().add(btnOK, null);
this.getContentPane().add(btnCancel, null);
this.getContentPane().add(lblAttribute,null);
if (test.window.isDesign()){
 this.getContentPane().add(cmbAttribute, null);
}
else {
 this.getContentPane().add(txtAttribute, null);
}

this.setBounds(100,100,210,170);
this.show(true);}
```

```
/*              Set Methods                */
/**
 * Sets the first relation of the operator
 * @parameter relation1 (String)
**/
public void setRelation1 (String relation1){
 this.relation1 = relation1;
}

/**
 * Sets the second relation of the operator
 * @parameter relation2 (String)
**/
public void setRelation2 (String relation2){
 this.relation2 = relation2;
}

/**
 * Sets the name of the operator
 * @parameter name (String)
**/
public void setName (String name){
 this.name = name;
}

/**
 * Sets the intersect attribute
 * @parameter attribute (String)
**/
public void setAttribute (String attribute){
 this.attribute = attribute;
}

/*              Get Methods                */
/**
 * Gets the number of the input nodes of the operator
**/
public int getnumberOfNodes(){
 return this.NO_INPUT_NODES;
}

/**
 * Gets the type of the operator
**/
public String getOperatorType(){
  return this.OPERATOR_TYPE;
 }

/**
  * Gets the first relationof the operator
**/
public String getRelation1 (){
  return this.relation1;
 }

/**
  * Gets the second relation of the operator
**/
public String getRelation2 (){
  return this.relation2;
 }

/**
  * Gets the name of the operator
**/
public String getName (){
  return this.name;
 }

/**
  * Gets the intersect attribute of the operator
**/
public String getAttribute (){
  return this.attribute;
 }

/*              WINDOW ACTIVATIONS                    */
/**
  * The window listener methods
  * @param event (WindowEvent)
**/
public void windowClosing        (WindowEvent event){System.exit(0);}
public void windowActivated      (WindowEvent event){}
public void windowClosed         (WindowEvent event){}
public void windowDeactivated    (WindowEvent event){}
public void windowDeiconified    (WindowEvent event){}
public void windowIconified      (WindowEvent event){}
public void windowOpened         (WindowEvent event){}
```

```
/**
  * button okay action performed
  * Takes the information entered for the operator. Creates a new operator and
  * calls add or change method of the draw class. And sends new operator
  * information there to be added to the list.
  * @param e (ActionEvent)
**/
void btnOK_actionPerformed(ActionEvent e) {
 if(chkResult.isSelected()){
  display = true;
 }
 else{
  display = false;
 }
 setName(txtName.getText());
 setRelation1((String)cmbRelation1.getSelectedItem());
 setRelation2((String)cmbRelation2.getSelectedItem());
 if (test.window.isDesign()){
  setAttribute((String)cmbAttribute.getSelectedItem());
 }
 else{
  setAttribute(txtAttribute.getText());
  if (txtAttribute.getText().equals("")){
   test.window.message.setMessage("You did not enter the attribute list");
   test.window.message.show(true);
  }
 }

 String [] input = {getRelation1(),getRelation2(),getAttribute(),"none","none"};
 buildQuery();
 if (test.window.isDesign()){
  test.window.designQueries.addElement(query);
 }
 else {
  if (whoIsCalling){
    test.window.cnvDFQL.change
                    (current,indexNum,x,y,getName(),getnumberOfNodes(),
                     input,display, getOperatorType(), query);
  }
  else {
    test.window.cnvDFQL.add
                    (x,y,getName(),getnumberOfNodes(),input,display,
                     getOperatorType(), query);
  }}
 this.dispose();
}
```

```
/**
  * Loads the relations with the tables of the selected database and previously
  * created operators.
**/
void loadSelect() {

  if(!whoIsCalling){
   txtName.setText("Intersect"+test.window.Counter[12]++);
  }

  Vector deneme = test.window.currentDatabase.getTableNames();
  for (int index = 0 ; index < deneme.size(); index++) {
    cmbRelation1.addItem((String) deneme.elementAt(index));
    cmbRelation2.addItem((String) deneme.elementAt(index));
  }

  for (int xi = 0; xi <test.window.cnvDFQL.numOfOperators;xi++){
   cmbRelation1.addItem(test.window.cnvDFQL.operators[xi].name);
   cmbRelation2.addItem(test.window.cnvDFQL.operators[xi].name);
  }
  cmbRelation1.removeItem(name);
  cmbRelation2.removeItem(name);

  if (relation1== null)
    cmbRelation1.setSelectedIndex(0);
  else
    cmbRelation1.setSelectedItem(relation1);
  if (relation2== null)
    cmbRelation2.setSelectedIndex(0);
  else
    cmbRelation2.setSelectedItem(relation2);
  txtAttribute.setText(attribute);
}

/**
  * button cancel action performed
  * terminates this frame without creating an operator. decrements the counter
  * @param e (ActionEvent)
**/
void btnCancel_actionPerformed(ActionEvent e) {
   test.window.Counter[12]--;
   this.dispose();
}
```

```
/**
 * This portion of the code produces the SQL query for this operator according
 * to the information entered by user.
**/
public void buildQuery(){
  String checkRelation = isOtherOperator (relation1);
     if (checkRelation == null) {
     query = "select distinct " + attribute + " from " + relation1 +" S where S."
              + attribute + " in";
  }
  else {
     query = "select distinct " + attribute + " from " + "("
              +checkRelation + ") S where S." + attribute + " in";
  }
  checkRelation = isOtherOperator(relation2);

  if (checkRelation == null) {
     query = query + " (select " + attribute + " from " + relation2 + ")";
  }
  else {
     query = query + "(select " + attribute + " from " + "(" +checkRelation + "))";
  }
}

/**
 * This portion of the code looks for the type of the relations, if they are
 * tables it returns null, otherwise if is another operator, it returns the
 * query of the operator which the current operator related to.
**/
public String isOtherOperator (String type) {
  String result = null;
  for (int index = 0; index < test.window.DFQLQuery.size(); index++) {
    Operator temp = (Operator) test.window.DFQLQuery.elementAt(index);
    if (type.equalsIgnoreCase(temp.name)) {
      result = temp.query;
      break;
    }
  }
return result;
}
```

```
    /**
     * This function is used to fill the combo boxes with the selection alternatives
     * if this operator is created as a part of a user defined operator then the
     * relation and condition combo boxes are filled with user defined operator
     * input items.
    **/
    private void loadDesign(){
      txtName.setText(name);
      for (int index = 0 ; index < test.window.designRelation.size(); index++) {
        cmbRelation1.addItem((String) test.window.designRelation.elementAt
                                (index));
        cmbRelation2.addItem((String) test.window.designRelation.elementAt
                                (index));
      }

      for (int opIndex = 0 ; opIndex < test.window.designOps.size(); opIndex++) {
        if (!name.equalsIgnoreCase((String) test.window.designOps.elementAt
                                    (opIndex))){
          cmbRelation1.addItem((String) test.window.designOps.elementAt
                                  (opIndex));
          cmbRelation2.addItem((String) test.window.designOps.elementAt
                                  (opIndex));
        }
      }

      for (int attIndex = 0; attIndex < test.window.designAttribute.size();
            attIndex++){
        cmbAttribute.addItem((String) test.window.designAttribute.elementAt
                                (attIndex));
      }
    }
  }
```

22. OperatorJoin.Java

```
package ThesisGUI;

import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;
```

```
/**
  * This frame is used to allow the user to enter the operator information
  * for join operator. It also allwos user to change the information
  * previously entered.
  * AUTHOR : IS & BA
**/
public class OperatorJoin extends JFrame implements WindowListener{

/** Type of the operator (constant) **/
 final static String OPERATOR_TYPE = "Join";

 /** Input node number of the operator (constant) **/
 final static int NO_INPUT_NODES = 3;

 /** X coordinate of the operator**/
 private int x;

 /** Y coordinate of the operator**/
 private int y;

 /** name of the operator**/
 private String name;

 /** first input node relation**/
 private String relation1;

 /** second input node relation**/
 private String relation2;

 /** Condition**/
 private String condition;

 /** display information for the operator**/
 private boolean display = false;

 /** SQL query of the operator**/
 private String query;

 /** Specify why this frame is called (to create a new one (false)
   * or change the current operator (true)**/
 private boolean whoIsCalling = false;

 /** The index of the operator in the Operators list **/
 private int indexNum;
```

```
/** The index of the operator in the DFQLList**/
private int current;

/** The list of the possible index of the operator in the DFQLList**/
private String [] operators = {"=", ">", ">=", "<", "<="} ;

/** name label of the frame**/
JLabel lblName        = new JLabel();

/** first relation label of the frame**/
JLabel lblRelation1    = new JLabel();

/** second relation label of the frame**/
JLabel lblRelation2    = new JLabel();

/** text area to enter the name of the operator default diff#**/
JTextField txtName     = new JTextField();

/** Okay button to confirm the info entered**/
JButton btnOK          = new JButton();

/** check box for display information**/
JCheckBox chkResult    = new JCheckBox();

/** Combo box lists the available operators or table names that can be
 * related to this operator**/
JComboBox cmbRelation1  = new JComboBox();

/** Combo box lists the available operators or table names that can be
 * related to this operator**/
JComboBox cmbRelation2  = new JComboBox();

/** join attribute list label of the frame**/
JLabel lblCondition = new JLabel();

/** text area to enter the condition for the operator**/
JTextField txtCondition = new JTextField();

/** Combo box lists the available conditions that can be
 * related to this operator**/
JComboBox cmbDesignCondition    = new JComboBox();

/** Button cancel to cancel this operation**/
JButton btnCancel      = new JButton();
```

```
/** Combo box lists the available operators for join operation **/
JComboBox cmbOperator = new JComboBox();

/*           CONSTRUCTOR                    */
/**
 * Default constructor
 * @parameters X and Y (integer) coordinates of the operator
**/
public OperatorJoin(int x, int y) {
 try {
  this.x = x;
  this.y = y;
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}

/**
 * Constructor with all parameters
 * @parameters @ parameters x, y, caller1, caller2 (integer),
 * name,relation1, relation2, condition        (String),
 * display                               (boolean)
**/
public OperatorJoin(int x, int y, int caller1, int caller2, String name,
           String relation1, String relation2, String condition,
           boolean display){
 this.x             = x;
 this.y             = y;
 this.current       = caller1;
 this.indexNum      = caller2;
 this.whoIsCalling  = true;
 this.name          = name;
 this.relation1     = relation1;
 this.relation2     = relation2;
 int length         = condition.length();
 StringBuffer sub   = new StringBuffer(condition);
 condition          = sub.substring(0,length-2);
 this.condition     = condition;
 this.display       = display;

 txtName.setText         (name);
 txtCondition.setText    (condition);
 if (display){
  chkResult.setSelected(true);
  }
  else{
   chkResult.setSelected(false);
  }

  try  {
   jbInit();
  }
   catch(Exception e) {
    e.printStackTrace();
  }
}

/**
  * Design constructor
  * @parameters name
**/
public OperatorJoin(String name) {
  try  {
   this.name = name;
   jbInit();
  }
   catch(Exception e) {
    e.printStackTrace();
  }
}

/*              INITIATE METHOD                    */
/**
  * The initiation
  * creates the frame for user to enter the operator data, and adds
  * buttons, labels, text area and combo boxes.
**/
 private void jbInit() throws Exception {
 lblName.setText              ("Name");
 this.setTitle                ("Join");
 lblRelation1.setText         ("Relation1");
 lblRelation2.setText         ("Relation2");
 lblCondition.setText         ("Condition");
 btnOK.setText                ("OK");
 btnCancel.setText            ("Cancel");
 chkResult.setText            ("Display");

 btnCancel.setBounds                (new Rectangle(78, 93, 60, 23));
 cmbOperator.setBounds              (new Rectangle(150, 68, 44, 17));
 lblName.setBounds                  (new Rectangle(4, 2, 93, 18));
```

```
lblRelation1.setBounds           (new Rectangle(4, 22, 93, 18));
lblRelation2.setBounds           (new Rectangle(5, 45, 93, 18));
lblCondition.setBounds           (new Rectangle(4, 67, 93, 18));
txtName.setBounds                (new Rectangle(80, 2, 114, 17));
txtCondition.setBounds           (new Rectangle(80, 68, 59, 17));
cmbDesignCondition.setBounds     (new Rectangle(80, 68, 59, 17));
btnOK.setBounds                  (new Rectangle(147, 93, 53, 22));
chkResult.setBounds              (new Rectangle(4, 93, 64, 22));
cmbRelation1.setBounds           (new Rectangle(80, 23, 114, 17));
cmbRelation2.setBounds           (new Rectangle(80, 46, 114, 17));

for (int ix = 0 ; ix < 5; ix++){
  cmbOperator.addItem(operators[ix]);
}

if (test.window.isDesign()){
 loadDesign();
}
else {
 loadSelect();
}

btnOK.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnOK_actionPerformed(e);
 }});

btnCancel.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnCancel_actionPerformed(e);
 }});

cmbRelation1.setBorder          (null);
chkResult.setBorderPainted      (true);
cmbRelation2.setBorder          (null);

chkResult.setFont  (new java.awt.Font("Dialog", 0, 11));
btnOK.setFont      (new java.awt.Font("Dialog", 0, 11));
btnCancel.setFont  (new java.awt.Font("Dialog", 0, 11));

btnCancel.setBorder        (BorderFactory.createRaisedBevelBorder());
chkResult.setBorder        (BorderFactory.createRaisedBevelBorder());
btnOK.setBorder            (BorderFactory.createRaisedBevelBorder());

this.getContentPane().setLayout(null);
this.getContentPane().add(lblName, null);
```

```
  this.getContentPane().add(txtName, null);
  this.getContentPane().add(lblRelation1, null);
  this.getContentPane().add(cmbRelation2, null);
  this.getContentPane().add(lblRelation2, null);
  if (test.window.isDesign()){
   this.getContentPane().add(cmbDesignCondition, null);
  }
  else{
   this.getContentPane().add(txtCondition, null);
  }

  this.getContentPane().add(lblCondition, null);
  this.getContentPane().add(chkResult, null);
  this.getContentPane().add(btnCancel, null);
  this.getContentPane().add(btnOK, null);
  this.getContentPane().add(cmbRelation1, null);
  this.getContentPane().add(cmbOperator, null);

  this.setBounds(100,100,210,150);
  this.show(true);
 }

 /*              Set Methods                */
 /**
  * Sets the first relation of the operator
  * @parameter relation1 (String)
 **/
 public void setRelation1 (String relation1){
  this.relation1 = relation1;
 }

 /**
  * Sets the second relation of the operator
  * @parameter relation2 (String)
 **/
 public void setRelation2 (String relation2){
  this.relation2 = relation2;
 }

 /**
  * Sets the name of the operator
  * @parameter name (String)
 **/
 public void setName (String name){
  this.name = name;
 }
```

```
/**
  * Sets the condition of the operator
  * @parameter condition (String)
**/
public void setCondition (String condition){
 this.condition = condition;
}

/*              Get Methods                    */
/**
  * Gets the number of the input nodes of the operator
**/
public int getnumberOfNodes(){
 return this.NO_INPUT_NODES;
}

/**
  * Gets the type of the operator
**/
public String getOperatorType(){
 return this.OPERATOR_TYPE;
}

/**
  * Gets the first relationof the operator
**/
public String getRelation1 (){
 return this.relation1;
}

/**
  * Gets the second relation of the operator
**/
public String getRelation2 (){
 return this.relation2;
}

/**
  * Gets the name of the operator
**/
public String getName (){
 return this.name;
}
```

```
/**
 * Gets the condition of the operator
**/
public String getCondition (){
 return this.condition;
}

/*              WINDOW ACTIVATIONS                    */
/**
 * The window listener methods
 * @param event (WindowEvent)
**/
public void windowClosing          (WindowEvent event){System.exit(0);}
public void windowActivated        (WindowEvent event){}
public void windowClosed           (WindowEvent event){}
public void windowDeactivated      (WindowEvent event){}
public void windowDeiconified      (WindowEvent event){}
public void windowIconified        (WindowEvent event){}
public void windowOpened           (WindowEvent event){}

/**
 * button okay action performed
 * Takes the information entered for the operator. Creates a new operator and
 * calls add or change method of the draw class. And sends new operator
 * information there to be added to the list.
 * @param e (ActionEvent)
**/
void btnOK_actionPerformed(ActionEvent e) {
 if(chkResult.isSelected()){
   display = true;
 }
 else{
  display = false;
 }
 setName(txtName.getText());
 setRelation1((String)cmbRelation1.getSelectedItem());
 setRelation2((String)cmbRelation2.getSelectedItem());

 if (test.window.isDesign()){
   setCondition((String)cmbDesignCondition.getSelectedItem()+"
                   "+cmbOperator.getSelectedItem());
 }
 else {
  if (txtCondition.getText().equals("")){
    test.window.message.setMessage
                  ("You did not enter the join condition for the relations");
    test.window.message.show(true);
   }
   else {
    String cond = txtCondition.getText()+" "+cmbOperator.getSelectedItem();
    setCondition(cond);
   }
  }

  String [] input = {getRelation1(),getRelation2(),getCondition(),"none"};
  buildQuery();

  if (test.window.isDesign()){
   System.out.println(query);
   test.window.designQueries.addElement(query);
  }
  else {
   if (whoIsCalling){
     test.window.cnvDFQL.change
                    (current,indexNum,x,y,getName(),getnumberOfNodes(),
                     input,display, getOperatorType(), query);
   }
   else {
     test.window.cnvDFQL.add
                    (x,y,getName(),getnumberOfNodes(),input,display,
                     getOperatorType(), query);
   }
  }
  this.dispose();
 }

/**
  * Loads the relations with the tables of the selected database and previously
  * created operators.
**/
void loadSelect() {

 Vector deneme = test.window.currentDatabase.getTableNames();

  if(!whoIsCalling){
   txtName.setText("Join"+test.window.Counter[2]++);
  }

  for (int index = 0 ; index < deneme.size(); index++) {
    cmbRelation1.addItem((String) deneme.elementAt(index));
    cmbRelation2.addItem((String) deneme.elementAt(index));
  }
  for (int xi = 0; xi <test.window.cnvDFQL.numOfOperators;xi++) {
   cmbRelation1.addItem(test.window.cnvDFQL.operators[xi].name);
   cmbRelation2.addItem(test.window.cnvDFQL.operators[xi].name);
  }
  cmbRelation1.removeItem(name);
  cmbRelation2.removeItem(name);

  if (relation1== null)
    cmbRelation1.setSelectedIndex(0);
  else
    cmbRelation1.setSelectedItem(relation1);
  if (relation2== null)
    cmbRelation2.setSelectedIndex(0);
  else
    cmbRelation2.setSelectedItem(relation2);
}

/**
  * button cancel action performed
  * terminates this frame without creating an operator. decrements the counter
  * @param e (ActionEvent)
**/
void btnCancel_actionPerformed(ActionEvent e) {
   test.window.Counter[2]--;
   this.dispose();
}

/**
  * This portion of the code produces the SQL query for this operator according
  * to the information entered by user.
**/
public void buildQuery (){
  String checkRelation = isOtherOperator (relation1);
  if (checkRelation == null) {
     query = "select distinct * from " + relation1+"
               j"+test.window.Counter[2]*2+",";
  }
  else {
     query = "select distinct * from " + "(" +checkRelation + ")
               j"+test.window.Counter[2]*2+",";
  }
  checkRelation = isOtherOperator(relation2);

  if (checkRelation == null) {
     query = query + relation2 + " j"+test.window.Counter[2]*2 +1 ;
  }
  else {
    query = query + "(" +checkRelation + ") j"+test.window.Counter[2]*2 +1;
  }

  if (!(condition.equals(""))) {

    StringBuffer sub          = new StringBuffer (condition);
    int length                = condition.length();
    String cond               = sub.substring(0,length-2);
    String oprt               = sub.substring(length-2);

    query = query + " where j"+test.window.Counter[2]*2+"." + cond +" " + oprt
               + " j"+test.window.Counter[2]*2+1+"." + cond;
  }
}

/**
 * This portion of the code looks for the type of the relations, if they are
 * tables it returns null, otherwise if is another operator, it returns the
 * query of the operator which the current operator related to.
**/
public String isOtherOperator (String type) {
  String result = null;
  for (int index = 0; index < test.window.DFQLQuery.size(); index++) {
    Operator temp = (Operator) test.window.DFQLQuery.elementAt(index);
    if (type.equalsIgnoreCase(temp.name)) {
      result = temp.query;
      break;
    }
  }
  return result;
}

/**
 * This function is used to fill the combo boxes with the selection alternatives
 * if this operator is created as a part of a user defined operator then the
 * relation and condition combo boxes are filled with user defined operator
 * input items.
**/
private void loadDesign(){
  txtName.setText(name);
  for (int index = 0 ; index < test.window.designRelation.size(); index++) {
    cmbRelation1.addItem((String) test.window.designRelation.elementAt
                          (index));
    cmbRelation2.addItem((String) test.window.designRelation.elementAt
                                    (index));
    }

    for (int opIndex = 0 ; opIndex < test.window.designOps.size(); opIndex++) {
      if (!name.equalsIgnoreCase((String) test.window.designOps.elementAt
                                          (opIndex))){
        cmbRelation1.addItem((String) test.window.designOps.elementAt
                                  (opIndex));
        cmbRelation2.addItem((String) test.window.designOps.elementAt
                                  (opIndex));
      }
    }

    for (int conIndex = 0; conIndex < test.window.designCondition.size();
          conIndex++){
      cmbDesignCondition.addItem
              ((String) test.window.designCondition.elementAt(conIndex));
    }
  }
}
```

23. OperatorProject.Java

```
package ThesisGUI;

import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;

/**
 * This frame is used to allow the user to enter the operator information
 * for project operator. It also allwos user to change the information
 * previously entered.
 * AUTHOR : IS & BA
**/
public class OperatorProject extends JFrame implements WindowListener{

  /** Type of the operator (constant) **/
  final static String OPERATOR_TYPE = "Project";
```

```
/** Input node number of the operator (constant) **/
final static int NO_INPUT_NODES = 2;

/** X coordinate of the operator**/
private int x;

/** Y coordinate of the operator**/
private int y;

/** name of the operator**/
private String name;

/** input node relation**/
private String relation;

/** input node attribute list**/
private String attributeList;

/** display information for the operator**/
private boolean display = false;

/** SQL query of the operator**/
private String query;

/** Specify why this frame is called (to create a new one (false)
  * or change the current operator (true)**/
private boolean whoIsCalling = false;

/** The index of the operator in the Operators list **/
private int indexNum;

/** The index of the operator in the DFQLList**/
private int current;

/** name label of the frame**/
JLabel lblName          = new JLabel();

/** first relation label of the frame**/
JLabel lblRelation      = new JLabel();

/** second relation label of the frame**/
JLabel lblAttributeList     = new JLabel();

/** text area to enter the name of the operator default diff#**/
JTextField txtName      = new JTextField();
```

```
/** Okay button to confirm the info entered**/
JButton btnOK         = new JButton();

/** check box for display information**/
JCheckBox chkResult    = new JCheckBox();

/** Combo box lists the available operators or table names that can be
 * related to this operator**/
JComboBox cmbRelation  = new JComboBox();

/** Allows user to enter the attribute list**/
JTextField txtAttributeList = new JTextField();

/** Combo box lists the available attributes that can be
 * related to this operator**/
JComboBox cmbAttribute    = new JComboBox();

/** Button cancel to cancel this operation**/
JButton btnCancel      = new JButton();

/*            CONSTRUCTOR                  */
/**
 * Default constructor
 * @parameters X and Y (integer) coordinates of the operator
**/
public OperatorProject(int x, int y) {
 try  {
  this.x = x;
  this.y = y;
  this.whoIsCalling = false;
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}

/**
 * Constructor with all parameters
 * @parameters @ parameters x, y, caller1, caller2 (integer),
 * name,relation, attList                   (String),
 * display                              (boolean)
**/
public OperatorProject(int x, int y, int caller1, int caller2, String name,
               String relation, String attList, boolean display){
 this.x              = x;
```

```
    this.y              = y;
    this.current        = caller1;
    this.indexNum       = caller2;
    this.whoIsCalling   = true;
    this.name           = name;
    this.relation       = relation;
    this.attributeList  = attList;
    this.display        = display;

    txtName.setText           (name);
    txtAttributeList.setText  (attributeList);

    if (display){
      chkResult.setSelected(true);
    }
    else{
      chkResult.setSelected(false);
    }

    try {
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  /**
   * Design constructor
   * @parameters name
  **/
  public OperatorProject(String name) {
    try {
      this.name = name;
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }
```

```
/*              INITIATE METHOD                */
/**
 * The initiation
 * creates the frame for user to enter the operator data, and adds
 * buttons, labels, text area and combo boxes.
**/
private void jbInit() throws Exception {
 lblName.setText              ("Name");
 this.setTitle                ("Project");
 lblRelation.setText          ("Relation");
 lblAttributeList.setText     ("Att. List");
 btnOK.setText                ("OK");
 chkResult.setText            ("Display");
 btnCancel.setText            ("Cancel");

 lblName.setBounds            (new Rectangle(4, 2, 93, 18));
 lblRelation.setBounds        (new Rectangle(4, 27, 93, 18));
 lblAttributeList.setBounds (new Rectangle(3, 49, 93, 18));
 txtName.setBounds            (new Rectangle(71, 3, 114, 17));
 txtAttributeList.setBounds (new Rectangle(70, 50, 114, 17));
 cmbAttribute.setBounds       (new Rectangle(70, 50, 114, 17));
 btnOK.setBounds              (new Rectangle(136, 73, 51, 23));
 cmbRelation.setBounds        (new Rectangle(71, 28, 114, 17));
 btnCancel.setBounds          (new Rectangle(70, 73, 60, 23));
 chkResult.setBounds          (new Rectangle(2, 73, 62, 23));

 if (test.window.isDesign()){
  loadDesign();
 }
 else {
  loadSelect();
 }

 btnOK.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   btnOK_actionPerformed(e);
  }});

 btnCancel.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   btnCancel_actionPerformed(e);
  }});

 chkResult.setBorderPainted        (true);
 cmbRelation.setBorder             (null);
```

```
  chkResult.setFont   (new java.awt.Font("Dialog", 0, 11));
  btnCancel.setFont   (new java.awt.Font("Dialog", 0, 11));
  btnOK.setFont       (new java.awt.Font("Dialog", 0, 11));

  btnOK.setBorder            (BorderFactory.createRaisedBevelBorder());
  btnCancel.setBorder        (BorderFactory.createRaisedBevelBorder());
  chkResult.setBorder        (BorderFactory.createRaisedBevelBorder());

  this.getContentPane().setLayout(null);
  this.getContentPane().add(lblName, null);
  this.getContentPane().add(lblRelation, null);
  this.getContentPane().add(cmbRelation, null);
  this.getContentPane().add(txtName, null);
  this.getContentPane().add(btnCancel, null);
  this.getContentPane().add(lblAttributeList, null);
  if (test.window.isDesign()){
   this.getContentPane().add(cmbAttribute, null);
  }
  else {
   this.getContentPane().add(txtAttributeList, null);
  }

  this.getContentPane().add(chkResult, null);
  this.getContentPane().add(btnOK, null);

  this.setBounds(100,100,200,130);
  this.show(true);
 }

/*            Set Methods                 */
/**
  * Sets the relation of the operator
  * @parameter relation (String)
**/
public void setRelation (String relation){
 this.relation = relation;
}

/**
  * Sets the attribute list of the operator
  * @parameter attributeList (String)
**/
public void setAttList (String attributeList){
 this.attributeList = attributeList;
}
```

```
/**
 * Sets the name of the operator
 * @parameter name (String)
**/
public void setName (String name){
 this.name = name;
}

/*          Get Methods          */
/**
 * Gets the input node number of the operator
**/
public int getnumberOfNodes(){
 return this.NO_INPUT_NODES;
}

/**
 * Gets the type of the operator
**/
public String getOperatorType(){
 return this.OPERATOR_TYPE;
}

/**
 * Gets the relation of the operator
**/
public String getRelation (){
 return this.relation;
}

/**
 * Gets the attribute list of the operator
**/
public String getAttList (){
 return this.attributeList;
}

/**
 * Gets the name of the operator
**/
public String getName (){
 return this.name;
}
```

```
/*            WINDOW ACTIVATIONS                    */
/**
 * The window listener methods
 * @param event (WindowEvent)
**/
public void windowClosing        (WindowEvent event){System.exit(0);}
public void windowActivated      (WindowEvent event){}
public void windowClosed         (WindowEvent event){}
public void windowDeactivated    (WindowEvent event){}
public void windowDeiconified    (WindowEvent event){}
public void windowIconified      (WindowEvent event){}
public void windowOpened         (WindowEvent event){}

/**
 * button okay action performed
 * Takes the information entered for the operator. Creates a new operator and
 * calls add or change method of the draw class. And sends new operator
 * information there to be added to the list.
 * @param e (ActionEvent)
**/
void btnOK_actionPerformed(ActionEvent e) {
 if (chkResult.isSelected()){
  display = true;
 }
 else{
  display = false;
 }
 setName     (txtName.getText());
 setRelation((String)cmbRelation.getSelectedItem());
 if (test.window.isDesign()){
  setAttList((String)cmbAttribute.getSelectedItem());
 }
 else{
  setAttList(txtAttributeList.getText());
  if (txtAttributeList.getText().equals("")){
   test.window.message.setMessage("You did not enter the attribute list");
   test.window.message.show(true);
  }
 }
 String [] input = {getRelation(),getAttList(),"none","none"};

 buildQuery();
 if (test.window.isDesign()){
  test.window.designQueries.addElement(query);
 }
 else {
   if (whoIsCalling) {
    test.window.cnvDFQL.change
                          (current,indexNum,x,y,getName(),getnumberOfNodes(),
                           input,display, getOperatorType(),query);
   }
   else {
    test.window.cnvDFQL.add
                          (x,y,getName(),getnumberOfNodes(),input,display,
                           getOperatorType(), query);
   }
  }
  this.dispose();
 }

 /**
  * Loads the relations with the tables of the selected database and previously
  * created operators.
 **/
 void loadSelect() {
  Vector deneme = test.window.currentDatabase.getTableNames();

  if(!whoIsCalling){
   txtName.setText("Project"+test.window.Counter[1]++);
  }

  for (int index = 0 ; index < deneme.size(); index++) {
    cmbRelation.addItem((String) deneme.elementAt(index));
  }

  for (int xi = 0; xi <test.window.cnvDFQL.numOfOperators;xi++)
   cmbRelation.addItem(test.window.cnvDFQL.operators[xi].name);

  cmbRelation.removeItem(name);
  if (relation == null)
    cmbRelation.setSelectedIndex(0);
  else
    cmbRelation.setSelectedItem(relation);
}

/**
  * button cancel action performed
  * terminates this frame without creating an operator. decrements the counter
  * @param e (ActionEvent)
 **/
void btnCancel_actionPerformed(ActionEvent e) {
    test.window.Counter[1]--;
    this.dispose();
  }

  /**
   * This portion of the code produces the SQL query for this operator according
   * to the information entered by user.
  **/
  public void buildQuery (){
    String checkRelation = isOtherOperator (relation);
    if (!attributeList.equals("") ){
      query = "select distinct " + attributeList;
    }
    else {
      query = "select distinct *";
    }

    if (checkRelation == null) {
       query = query + " from " + relation;
    }
    else {
       query = query + " from " + "(" +checkRelation + ")";
    }
  }

  /**
   * This portion of the code looks for the type of the relations, if they are
   * tables it returns null, otherwise if is another operator, it returns the
   * query of the operator which the current operator related to.
  **/
  public String isOtherOperator (String type) {
    String result = null;
    for (int index = 0; index < test.window.DFQLQuery.size(); index++) {
      Operator temp = (Operator) test.window.DFQLQuery.elementAt(index);
      if (type.equalsIgnoreCase(temp.name)) {
        result = temp.query;
        break;
      }
    }
  return result;
  }

  /**
   * This function is used to fill the combo boxes with the selection alternatives
   * if this operator is created as a part of a user defined operator then the
   * relation and condition combo boxes are filled with user defined operator
  * input items.
 **/
 private void loadDesign(){
   txtName.setText(name);
   for (int index = 0 ; index < test.window.designRelation.size(); index++) {
     cmbRelation.addItem((String) test.window.designRelation.elementAt
                                  (index));
   }

   for (int opIndex = 0 ; opIndex < test.window.designOps.size(); opIndex++) {
     if (!name.equalsIgnoreCase((String) test.window.designOps.elementAt
                                          (opIndex)))
       cmbRelation.addItem((String) test.window.designOps.elementAt
                                      (opIndex));
   }

   for (int attIndex = 0; attIndex < test.window.designAttribute.size();
        attIndex++){
     cmbAttribute.addItem((String) test.window.designAttribute.elementAt
                                      (attIndex));
   }
 }
}
```

24. OperatorSelect.Java

```
package ThesisGUI;

import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;

/**
 * This frame is used to allow the user to enter the operator information
 * for select operator. It also allwos user to change the information
 * previously entered.
 * AUTHOR : IS & BA
**/
public class OperatorSelect extends JFrame implements WindowListener{
 /** Type of the operator (constant) **/
 final static String OPERATOR_TYPE = "Select";
```

```
/** Input node number of the operator (constant) **/
final static int NO_INPUT_NODES = 2;

/** X coordinate of the operator**/
private int x;

/** Y coordinate of the operator**/
private int y;

/** name of the operator**/
private String name;

/** input node relation**/
private String relation;

/** input node attribute list**/
private String condition;

/** display information for the operator**/
private boolean display = false;

/** SQL query of the operator**/
private String query;

/** Specify why this frame is called (to create a new one (false)
  * or change the current operator (true)**/
private boolean whoIsCalling = false;

/** The index of the operator in the Operators list **/
private int indexNum;

/** The index of the operator in the DFQLList**/
private int current;

/** name label of the frame**/
JLabel lblName          = new JLabel();

/** first relation label of the frame**/
JLabel lblRelation      = new JLabel();

/** second relation label of the frame**/
JLabel lblCondition     = new JLabel();

/** text area to enter the name of the operator default diff#**/
JTextField txtName      = new JTextField();
```

```
/** Okay button to confirm the info entered**/
JButton btnOK          = new JButton();

/** check box for display information**/
JCheckBox chkResult    = new JCheckBox();

/** Combo box lists the available operators or table names that can be
 * related to this operator**/
JComboBox cmbRelation  = new JComboBox();

/** Combo box lists the available conditions that can be
 * related to this operator**/
JComboBox cmbDesignCondition = new JComboBox();

/** Allows user to enter the attribute list**/
JTextField txtCondition = new JTextField();

/** Button cancel to cancel this operation**/
JButton btnCancel      = new JButton();

/*            CONSTRUCTOR                */

/**
 * Default constructor
 * @parameters X and Y (integer) coordinates of the operator
**/
public OperatorSelect(int x, int y) {
 try  {
  this.x = x;
  this.y = y;
  this.whoIsCalling = false;
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}

/**
 * Constructor with all parameters
 * @parameters @ parameters x, y, caller1, caller2 (integer),
 * name,relation, condition                    (String),
 * display                                    (boolean)
**/
public OperatorSelect(int x, int y, int caller1, int caller2, String name,
                    String relation, String condition, boolean display){
  this.x             = x;
  this.y             = y;
  this.current       = caller1;
  this.indexNum      = caller2;
  this.whoIsCalling  = true;
  this.name          = name;
  this.relation      = relation;
  this.condition     = condition;
  this.display       = display;

  txtName.setText          (name);
  txtCondition.setText     (condition);

  if (display){
   chkResult.setSelected(true);
  }
  else{
   chkResult.setSelected(false);
  }

  try  {
   jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
 }

/**
  * Design constructor
  * @parameters name
**/
public OperatorSelect(String name) {
 try  {
  this.name = name;
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}

/*           INITIATE METHOD              */
/**
 * The initiation
 * creates the frame for user to enter the operator data, and adds
 * buttons, labels, text area and combo boxes.
**/
private void jbInit() throws Exception {
 lblName.setText           ("Name");
 this.setTitle             ("Select");
 lblRelation.setText       ("Relation");
 lblCondition.setText      ("Condition");
 btnOK.setText             ("OK");
 btnCancel.setText         ("Cancel");
 chkResult.setText         ("Display");

 lblName.setBounds                (new Rectangle(4, 2, 93, 18));
 lblRelation.setBounds            (new Rectangle(4, 27, 93, 18));
 lblCondition.setBounds           (new Rectangle(2, 51, 93, 18));
 txtName.setBounds                (new Rectangle(71, 3, 114, 17));
 txtCondition.setBounds           (new Rectangle(70, 52, 114, 17));
 cmbDesignCondition.setBounds     (new Rectangle(70, 52, 114, 17));
 btnOK.setBounds                  (new Rectangle(138, 77, 49, 22));
 chkResult.setBounds              (new Rectangle(1, 77, 64, 22));
 btnCancel.setBounds              (new Rectangle(72, 77, 60, 22));
 cmbRelation.setBounds            (new Rectangle(71, 28, 114, 17));

 if (test.window.isDesign()){
  loadDesign();
 }
 else {
  loadSelect();
 }

 btnOK.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   btnOK_actionPerformed(e);
  }});

 btnCancel.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   btnCancel_actionPerformed(e);
  }});

 chkResult.setBorderPainted       (true);
 cmbRelation.setBorder            (null);

 chkResult.setFont  (new java.awt.Font("Dialog", 0, 11));
 btnOK.setFont      (new java.awt.Font("Dialog", 0, 11));
 btnCancel.setFont  (new java.awt.Font("Dialog", 0, 11));
```

```
  btnOK.setBorder            (BorderFactory.createRaisedBevelBorder());
  chkResult.setBorder        (BorderFactory.createRaisedBevelBorder());
  btnCancel.setBorder        (BorderFactory.createRaisedBevelBorder());

  this.getContentPane().setLayout(null);
  this.getContentPane().add(lblName, null);
  this.getContentPane().add(lblRelation, null);
  this.getContentPane().add(cmbRelation, null);
  this.getContentPane().add(txtName, null);
  this.getContentPane().add(lblCondition, null);
  if (test.window.isDesign()){
   this.getContentPane().add(cmbDesignCondition, null);
  }
  else{
   this.getContentPane().add(txtCondition, null);
  }
  this.getContentPane().add(btnCancel, null);
  this.getContentPane().add(chkResult, null);
  this.getContentPane().add(btnOK, null);

  this.setBounds(100,100,200,130);
  this.show(true);
 }

/*          Set Methods              */
/**
  * Sets the relation of the operator
  * @parameter relation (String)
**/
public void setRelation (String relation){
 .this.relation = relation;
}

/**
  * Sets the attribute list of the operator
  * @parameter attributeList (String)
**/
public void setCondition (String condition){
 this.condition = condition;
}

/**
  * Sets the name of the operator
  * @parameter name (String)
**/
```

```
public void setName (String name){
 this.name = name;
}

/*          Get Methods             */
/**
 * Gets the input node number of the operator
**/
public int getnumberOfNodes(){
 return this.NO_INPUT_NODES;
}

/**
 * Gets the type of the operator
**/
public String getOperatorType(){
 return this.OPERATOR_TYPE;
}

/**
 * Gets the relation of the operator
**/
public String getRelation (){
 return this.relation;
}

/**
 * Gets the attribute list of the operator
**/
public String getCondition (){
 return this.condition;
}

/**
 * Gets the name of the operator
**/
public String getName (){
 return this.name;
}

/*          WINDOW ACTIVATIONS              */
/**
 * The window listener methods
 * @param event (WindowEvent)
**/
public void windowClosing          (WindowEvent event){System.exit(0);}
public void windowActivated        (WindowEvent event){}
public void windowClosed           (WindowEvent event){}
public void windowDeactivated      (WindowEvent event){}
public void windowDeiconified      (WindowEvent event){}
public void windowIconified        (WindowEvent event){}
public void windowOpened           (WindowEvent event){}

/**
 * button okay action performed
 * Takes the information entered for the operator. Creates a new operator and
 * calls add or change method of the draw class. And sends new operator
 * information there to be added to the list.
 * @param e (ActionEvent)
**/
void btnOK_actionPerformed(ActionEvent e) {
 if (chkResult.isSelected()){
  display = true;
 }
 else{
  display = false;
 }
 setName(txtName.getText());
 setRelation((String)cmbRelation.getSelectedItem());
 if (test.window.isDesign()){
   setCondition((String)cmbDesignCondition.getSelectedItem());
 }
 else {
  setCondition(txtCondition.getText());
 }
 String [] input = {getRelation(),getCondition(),"none","none"};
 buildQuery();
 if (!test.window.isDesign()){
   if (whoIsCalling){
     test.window.cnvDFQL.change
                        (current,indexNum,x,y,getName(),getnumberOfNodes(),
                         input,display, getOperatorType(),query);
   }
   else{
     test.window.cnvDFQL.add
                        (x,y,getName(),getnumberOfNodes(),input,display,
                         getOperatorType(),query);
   }
 }
 else {
  test.window.designQueries.addElement(query);
 }
```

```
  this.dispose();
 }

 /**
  * Loads the relations with the tables of the selected database and previously
  * created operators.
 **/
 void loadSelect() {

  Vector deneme = test.window.currentDatabase.getTableNames();

  if(!whoIsCalling){
   txtName.setText("Select"+test.window.Counter[0]++);
  }

  for (int index = 0 ; index < deneme.size(); index++) {
    cmbRelation.addItem((String) deneme.elementAt(index));
  }

  for (int xi = 0; xi <test.window.cnvDFQL.numOfOperators;xi++)
   cmbRelation.addItem(test.window.cnvDFQL.operators[xi].name);

  cmbRelation.removeItem(name);
  if (relation == null)
    cmbRelation.setSelectedIndex(0);
  else
    cmbRelation.setSelectedItem(relation);
 }

 /**
  * This portion of the code produces the SQL query for this operator according
  * to the information entered by user.
 **/
 public void buildQuery(){
  String checkRelation = isOtherOperator (relation);
  if (checkRelation == null) {
    query = "select distinct * from " + relation;
  }
  else {
   query = "select distinct * from " + "(" +checkRelation + ")";
  }
  if (!(condition.equals(""))) {
    query = query + " where " + condition;
  }
 }
```

```
/**
 * button cancel action performed
 * terminates this frame without creating an operator. decrements the counter
 * @param e (ActionEvent)
**/
void btnCancel_actionPerformed(ActionEvent e) {
  test.window.Counter[0]--;
  this.dispose();
}

/**
 * This portion of the code looks for the type of the relations, if they are
 * tables it returns null, otherwise if is another operator, it returns the
 * query of the operator which the current operator related to.
**/
public String isOtherOperator (String type) {
  String result = null;
  for (int index = 0; index < test.window.DFQLQuery.size(); index++) {
    Operator temp = (Operator) test.window.DFQLQuery.elementAt(index);
    if (type.equalsIgnoreCase(temp.name)) {
      result = temp.query;
      break;
    }
  }
  return result;
}

/**
 * This function is used to fill the combo boxes with the selection alternatives
 * if this operator is created as a part of a user defined operator then the
 * relation and condition combo boxes are filled with user defined operator
 * input items.
**/
private void loadDesign(){
  txtName.setText(name);
  for (int index = 0 ; index < test.window.designRelation.size(); index++) {
    cmbRelation.addItem((String) test.window.designRelation.elementAt
                        (index));
  }

  for (int opIndex = 0 ; opIndex < test.window.designOps.size(); opIndex++) {
    if (!name.equalsIgnoreCase((String) test.window.designOps.elementAt
                               (opIndex)))
      cmbRelation.addItem((String) test.window.designOps.elementAt
                          (opIndex));
  }
```

```
    for (int conIndex = 0; conIndex < test.window.designCondition.size();
         conIndex++){
      cmbDesignCondition.addItem
                (String)test.window.designCondition.elementAt(conIndex));
    }
  }
}
```

25. OperatorUnion.Java

```
package ThesisGUI;

import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;

/**
 * This frame is used to allow the user to enter the operator information
 * for union operator. It also allwos user to change the information
 * previously entered.
 * AUTHOR : IS & BA
**/
public class OperatorUnion extends JFrame implements WindowListener{

 /** Type of the operator (constant) **/
 final static String OPERATOR_TYPE = "Union";

 /** Input node number of the operator (constant) **/
 final static int NO_INPUT_NODES = 2;

 /** X coordinate of the operator**/
 private int x;

 /** Y coordinate of the operator**/
 private int y;

 /** name of the operator**/
 private String name;
```

```
/** first input node relation**/
private String relation1;

/** second input node relation**/
private String relation2;

/** display information for the operator**/
private boolean display = false;

/** SQL query of the operator**/
private String query;

/** Specify why this frame is called (to create a new one (false)
  * or change the current operator (true)**/
private boolean whoIsCalling = false;

/** The index of the operator in the Operators list **/
private int indexNum;

/** The index of the operator in the DFQLList**/
private int current;

/** name label of the frame**/
JLabel lblName          = new JLabel();

/** first relation label of the frame**/
JLabel lblRelation1     = new JLabel();

/** second relation label of the frame**/
JLabel lblRelation2     = new JLabel();

/** text area to enter the name of the operator default diff#**/
JTextField txtName      = new JTextField();

/** Okay button to confirm the info entered**/
JButton btnOK           = new JButton();

/** check box for display information**/
JCheckBox chkResult     = new JCheckBox();

/** Combo box lists the available operators or table names that can be
  * related to this operator**/
JComboBox cmbRelation1  = new JComboBox();
```

```
/** Combo box lists the available operators or table names that can be
  * related to this operator**/
JComboBox cmbRelation2  = new JComboBox();

/** Button cancel to cancel this operation**/
JButton btnCancel       = new JButton();

/*              CONSTRUCTOR                     */
/**
  * Default constructor
  * @parameters X and Y (integer) coordinates of the operator
**/
public OperatorUnion(int x, int y) {
  try  {
    this.x = x;
    this.y = y;
    jbInit();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}

/**
  * Constructor with all parameters
  * @parameters @ parameters x, y, caller1, caller2 (integer),
  * name,relation1, relation2                    (String),
  * display                               (boolean)
**/
public OperatorUnion(int x, int y, int caller1, int caller2, String name,
              String relation1, String relation2, boolean display){
  this.x                = x;
  this.y                = y;
  this.current          = caller1;
  this.indexNum         = caller2;
  this.whoIsCalling     = true;
  this.name             = name;
  this.relation1        = relation1;
  this.relation2        = relation2;
  this.display          = display;

  txtName.setText      (name);

  if (display){
    chkResult.setSelected(true);
  }
  else{
   chkResult.setSelected(false);
  }

  try {
   jbInit();
  }
   catch(Exception e) {
    e.printStackTrace();
  }
 }

/**
  * Design constructor
  * @parameters name
**/
public OperatorUnion(String name) {
 try {
   this.name = name;
   jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
}

/*              INITIATE METHOD                    */
/**
  * The initiation
  * creates the frame for user to enter the operator data, and adds
  * buttons, labels, text area and combo boxes.
**/
private void jbInit() throws Exception {
 lblName.setText            ("Name");
 this.setTitle              ("Union");
 lblRelation1.setText       ("Relation1");
 lblRelation2.setText       ("Relation2");
 btnOK.setText              ("OK");
 btnCancel.setText          ("Cancel");
 chkResult.setText          ("Display");

 lblName.setBounds          (new Rectangle(4, 2, 93, 18));
 lblRelation1.setBounds     (new Rectangle(4, 27, 93, 18));
 lblRelation2.setBounds     (new Rectangle(4, 50, 93, 18));
 chkResult.setBounds        (new Rectangle(2, 78, 56, 22));
 cmbRelation1.setBounds     (new Rectangle(80, 28, 114, 17));
```

```
cmbRelation2.setBounds    (new Rectangle(80, 51, 114, 17));
btnCancel.setBounds       (new Rectangle(71, 78, 60, 23));
txtName.setBounds         (new Rectangle(80, 2, 114, 17));
btnOK.setBounds           (new Rectangle(143, 78, 52, 22));

if (test.window.isDesign()){
 loadDesign();
}
else {
 loadSelect();
}

btnOK.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnOK_actionPerformed(e);
 }});

btnCancel.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnCancel_actionPerformed(e);
 }});

chkResult.setBorderPainted      (true);
cmbRelation1.setBorder          (null);
cmbRelation2.setBorder          (null);

chkResult.setFont  (new java.awt.Font("Dialog", 0, 11));
btnCancel.setFont  (new java.awt.Font("Dialog", 0, 11));
btnOK.setFont      (new java.awt.Font("Dialog", 0, 11));

btnOK.setBorder           (BorderFactory.createRaisedBevelBorder());
chkResult.setBorder       (BorderFactory.createRaisedBevelBorder());
btnCancel.setBorder       (BorderFactory.createRaisedBevelBorder());

this.getContentPane().setLayout(null);
this.getContentPane().add(lblName, null);
this.getContentPane().add(txtName, null);
this.getContentPane().add(lblRelation1, null);
this.getContentPane().add(cmbRelation1, null);
this.getContentPane().add(cmbRelation2, null);
this.getContentPane().add(lblRelation2, null);
this.getContentPane().add(chkResult, null);
this.getContentPane().add(btnCancel, null);
this.getContentPane().add(btnOK, null);

this.setBounds(100,100,210,140);
```

```
 this.show(true);
}

/*          Set Methods               */
 /**
  * Sets the first relation of the operator
  * @parameter relation1 (String)
 **/
 public void setRelation1 (String relation1){
  this.relation1 = relation1;
 }

 /**
  * Sets the second relation of the operator
  * @parameter relation2 (String)
 **/
 public void setRelation2 (String relation2){
  this.relation2 = relation2;
 }

 /**
  * Sets the name of the operator
  * @parameter name (String)
 **/
 public void setName (String name){
  this.name = name;
 }

/*          Get Methods               */
 /**
  * Gets the number of the input nodes of the operator
 **/
 public int getnumberOfNodes(){
  return this.NO_INPUT_NODES;
 }

 /**
  * Gets the type of the operator
 **/
 public String getOperatorType(){
  return this.OPERATOR_TYPE;
 }
```

```
/**
  * Gets the first relationof the operator
 **/
 public String getRelation1 (){
  return this.relation1;
 }

/**
  * Gets the second relation of the operator
**/
public String getRelation2 (){
 return this.relation2;
}

/**
  * Gets the name of the operator
**/
public String getName (){
 return this.name;
}

/*              WINDOW ACTIVATIONS                    */
/**
  * The window listener methods
  * @param event (WindowEvent)
**/
public void windowClosing          (WindowEvent event){System.exit(0);}
public void windowActivated        (WindowEvent event){}
public void windowClosed           (WindowEvent event){}
public void windowDeactivated      (WindowEvent event){}
public void windowDeiconified      (WindowEvent event){}
public void windowIconified        (WindowEvent event){}
public void windowOpened           (WindowEvent event){}

/**
  * button okay action performed
  * Takes the information entered for the operator. Creates a new operator and
  * calls add or change method of the draw class. And sends new operator
  * information there to be added to the list.
  * @param e (ActionEvent)
**/
void btnOK_actionPerformed(ActionEvent e) {
 if (chkResult.isSelected()){
  display = true;
 }
 else{
    display = false;
   }
   setName(txtName.getText());
   setRelation1((String)cmbRelation1.getSelectedItem());
   setRelation2((String)cmbRelation2.getSelectedItem());
   String [] input = {getRelation1(),getRelation2(),"none","none"};
   buildQuery();
   if (!test.window.isDesign()){
    if (whoIsCalling){
     test.window.cnvDFQL.change
                    (current,indexNum,x,y,getName(),getnumberOfNodes(),
                     input,display, getOperatorType(), query);
    }
    else{
     test.window.cnvDFQL.add
                    (x,y,getName(),getnumberOfNodes(),input,display,
                     getOperatorType(), query);
    }
   }
   else {
    test.window.designQueries.addElement(query);
   }
   this.dispose();
  }

  /**
   * Loads the relations with the tables of the selected database and previously
   * created operators.
  **/
  void loadSelect() {

   if(!whoIsCalling){
    txtName.setText("Union"+test.window.Counter[3]++);
   }

   Vector deneme = test.window.currentDatabase.getTableNames();
   for (int index = 0 ; index < deneme.size(); index++) {
     cmbRelation1.addItem((String) deneme.elementAt(index));
     cmbRelation2.addItem((String) deneme.elementAt(index));
   }

   for (int xi = 0; xi <test.window.cnvDFQL.numOfOperators;xi++){
    cmbRelation1.addItem(test.window.cnvDFQL.operators[xi].name);
    cmbRelation2.addItem(test.window.cnvDFQL.operators[xi].name);
   }
  cmbRelation1.removeItem(name);
  cmbRelation2.removeItem(name);

  if (relation1== null)
    cmbRelation1.setSelectedIndex(0);
  else
    cmbRelation1.setSelectedItem(relation1);
  if (relation2== null)
    cmbRelation2.setSelectedIndex(0);
  else
    cmbRelation2.setSelectedItem(relation2);
}

/**
  * button cancel action performed
  * terminates this frame without creating an operator. decrements the counter
  * @param e (ActionEvent)
**/
void btnCancel_actionPerformed(ActionEvent e) {
   test.window.Counter[3]--;
   this.dispose();
}

/**
  * This portion of the code produces the SQL query for this operator according
  * to the information entered by user.
**/
public void buildQuery(){
  String checkRelation = isOtherOperator (relation1);
  if (checkRelation == null) {
     query = "select distinct * from " + relation1+" union ";
  }
  else {
     query = "select distinct * from " + "(" +checkRelation + ") union ";
  }
  checkRelation = isOtherOperator(relation2);

  if (checkRelation == null) {
     query = query + "select distinct * from " + relation2;
  }
  else {
     query = query + "select distinct * from " + "(" +checkRelation + ")";
  }
}
```

```
/**
  * This portion of the code looks for the type of the relations, if they are
  * tables it returns null, otherwise if is another operator, it returns the
  * query of the operator which the current operator related to.
 **/
 public String isOtherOperator (String type) {
   String result = null;
   for (int index = 0; index < test.window.DFQLQuery.size(); index++) {
     Operator temp = (Operator) test.window.DFQLQuery.elementAt(index);
     if (type.equalsIgnoreCase(temp.name)) {
       result = temp.query;
       break;
     }
   }
 return result;
 }

/**
  * This function is used to fill the combo boxes with the selection alternatives
  * if this operator is created as a part of a user defined operator then the
  * relation and condition combo boxes are filled with user defined operator
  * input items.
 **/
 private void loadDesign(){
   txtName.setText(name);
   for (int index = 0 ; index < test.window.designRelation.size(); index++) {
     cmbRelation1.addItem((String) test.window.designRelation.elementAt
                              (index));
     cmbRelation2.addItem((String) test.window.designRelation.elementAt
                              (index));
   }

   for (int opIndex = 0 ; opIndex < test.window.designOps.size(); opIndex++) {
     if (!name.equalsIgnoreCase((String) test.window.designOps.elementAt
                                   (opIndex))){
       cmbRelation1.addItem((String) test.window.designOps.elementAt
                              (opIndex));
       cmbRelation2.addItem((String) test.window.designOps.elementAt
                              (opIndex));
     }
   }
 }
}
```

26. OperatorUser.Java

```
package ThesisGUI;

import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.awt.*;
import javax.swing.*;
import java.util.Vector;
import javax.accessibility.*;
import javax.swing.UIManager;

/**
 * This frame is used to create user defined operators. User enters
 * the numbers of the relation, attribute and conditions of the new
 * operator and selects which operators are going to be used to create
 * the new one.
 * AUTHOR : IS & BA
**/
public class OperatorUser extends JFrame implements WindowListener {

 /** Type of the operator (constant) **/
  String operatorType;

  /** Input node number of the operator (constant) **/
  int noOfInputNodes;

  /** X coordinate of the operator**/
  private int x;

  /** Y coordinate of the operator**/
  private int y;

  /** name of the operator**/
  private String name;

  /** input node relation**/
  private String [] inputNodes = {"","","",""};

  /** display information for the operator**/
  private boolean display = false;

  /** SQL query of the operator**/
  private String querySkeleton;
```

```
/** SQL query of the operator**/
private String query;

/** name label of the frame**/
JLabel lblName          = new JLabel();

/** relation label of the frame**/
JLabel lblInput1      = new JLabel();

/** relation label of the frame**/
JLabel lblInput2      = new JLabel();

/** relation label of the frame**/
JLabel lblInput3      = new JLabel();

/** relation label of the frame**/
JLabel lblInput4      = new JLabel();

/** text area to enter the name of the operator default diff#**/
JTextField txtName      = new JTextField();

/** Okay button to confirm the info entered**/
JButton btnOK           = new JButton();

/** check box for display information**/
JCheckBox chkResult     = new JCheckBox();

/** Combo box lists the available operators or table names that can be
  * related to this operator**/
JComboBox cmbInput1  = new JComboBox();

/** Combo box lists the available operators or table names that can be
  * related to this operator**/
JComboBox cmbInput2  = new JComboBox();

/** Button cancel to cancel this operation**/
JButton btnCancel       = new JButton();

/** text area to enter the group attribute list of the operator **/
JTextField txtInput2 = new JTextField();

/** text area to enter the count attribute list of the operator **/
JTextField txtInput3 = new JTextField();

/** text area to enter the condition number of the operator **/
JTextField txtInput4 = new JTextField();
```

```
/** number of the relations **/
private int noOfRelations;

/** number of the conditions **/
private int noOfConditions;

/** number of the attributes **/
private int noOfAttributes;

/*              CONSTRUCTOR                    */
/**
 * Default constructor
 * @parameters X and Y (integer) coordinates of the operator
**/

public OperatorUser(int x, int y, int relation, int condition, int attribute,
            String skeleton, String type) {
 try  {
  this.x                    = x;
  this.y                    = y;
  this.noOfRelations        = relation;
  this.noOfConditions       = condition;
  this.noOfAttributes       = attribute;
  this.noOfInputNodes       = relation + condition + attribute;
  this.querySkeleton        = skeleton;
  this.operatorType         = type;
  jbInit();
 }
 catch(Exception e) {
  e.printStackTrace();
 }
}

/*              INITIATE METHOD                    */
/**
 * The initiation
 * creates the frame for user to enter the operator data, and adds
 * buttons, labels, text area and combo boxes.
**/
private void jbInit() throws Exception {
 Vector labels = new Vector();

 for (int relIndex = 1; relIndex <= noOfRelations; relIndex++){
  labels.addElement("Relation"+relIndex);
 }
```

```
for (int conIndex = 1; conIndex <= noOfConditions; conIndex++){
 labels.addElement("Condition"+conIndex);
}

for (int attIndex = 1; attIndex <= noOfAttributes; attIndex++){
 labels.addElement("Attribute"+attIndex);
}

btnOK.setText       ("OK");
btnCancel.setText   ("Cancel");
chkResult.setText   ("Display");
lblName.setText     ("Name");
this.setTitle       (operatorType);

lblName.setBounds          (new Rectangle(4, 2, 93, 18));
lblInput1.setBounds        (new Rectangle(3, 24, 93, 18));
lblInput2.setBounds        (new Rectangle(4, 47, 93, 18));
lblInput3.setBounds        (new Rectangle(4, 70, 93, 18));
lblInput4.setBounds        (new Rectangle(4, 95, 68, 16));

txtName.setBounds          (new Rectangle(80, 2, 114, 17));
txtInput2.setBounds        (new Rectangle(80, 48, 114, 17));
txtInput3.setBounds        (new Rectangle(80, 71, 114, 17));
txtInput4.setBounds        (new Rectangle(80, 95, 114, 17));
cmbInput1.setBounds        (new Rectangle(79, 25, 114, 17));
cmbInput2.setBounds        (new Rectangle(80, 48, 114, 17));

btnOK.setBounds            (new Rectangle(142, 117, 51, 22));
btnCancel.setBounds        (new Rectangle(73, 117, 60, 23));
chkResult.setBounds        (new Rectangle(5, 117, 58, 22));

txtName.setText     (operatorType);

loadSelect();

btnOK.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnOK_actionPerformed(e);
 }});

btnCancel.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  btnCancel_actionPerformed(e);
 }});
```

```
cmbInput1.setBorder             (null);
cmbInput2.setBorder             (null);
chkResult.setBorderPainted      (true);

chkResult.setFont  (new java.awt.Font("Dialog", 0, 11));
btnOK.setFont      (new java.awt.Font("Dialog", 0, 11));
btnCancel.setFont  (new java.awt.Font("Dialog", 0, 11));

btnOK.setBorder             (BorderFactory.createRaisedBevelBorder());
chkResult.setBorder         (BorderFactory.createRaisedBevelBorder());
btnCancel.setBorder         (BorderFactory.createRaisedBevelBorder());

this.getContentPane().setLayout(null);
this.getContentPane().add(lblName, null);
this.getContentPane().add(txtName, null);

if (noOfInputNodes > 3){
  lblInput4.setText   ((String)labels.elementAt(3));
  this.getContentPane().add(lblInput4, null);
  this.getContentPane().add(txtInput4, null);
}
if (noOfInputNodes > 2){
  lblInput3.setText   ((String)labels.elementAt(2));
  this.getContentPane().add(lblInput3, null);
  this.getContentPane().add(txtInput3, null);
}
if (noOfInputNodes > 1){
  lblInput2.setText   ((String)labels.elementAt(1));
  this.getContentPane().add(lblInput2, null);
  if (noOfRelations > 1) {
    this.getContentPane().add(cmbInput2, null);
  }
  else {
    this.getContentPane().add(txtInput2, null);
  }
}

if (noOfInputNodes > 0){
  lblInput1.setText   ((String)labels.elementAt(0));
  this.getContentPane().add(lblInput1, null);
  this.getContentPane().add(cmbInput1, null);
}

this.getContentPane().add(chkResult, null);
this.getContentPane().add(btnCancel, null);
this.getContentPane().add(btnOK, null);
```

```
  this.setBounds(100,100,210,180);
  this.show(true);
 }

 /*              Set Methods                  */
 /**
  * Sets the name of the operator
  * @parameter name (String)
 **/
 public void setName (String name){
  this.name = name;
 }

 /*              Get Methods                  */
 /**
  * gets the input number of the operator
 **/
 public int getnumberOfNodes(){
  return this.noOfInputNodes;
 }

 /**
  * gets the type of the operator
 **/
 public String getOperatorType(){
  return this.operatorType;
 }

 /**
  * gets the relation of the operator
 **/
 public String getInput1 (){
  return (String)cmbInput1.getSelectedItem();
 }

 /**
  * gets the group Attribute list of the operator
 **/
 public String getInput2 (){
  if (noOfRelations > 1){
    return (String)cmbInput2.getSelectedItem();
  }
  else {
    return txtInput2.getText();
  }}
```

```
/**
  * gets the condition of the operator
**/
public String getInput3 (){
  return txtInput3.getText();
}

/**
  * gets the condition of the operator
**/
public String getInput4 (){
  return txtInput4.getText();
}

/**
  * gets the name of the operator
**/
public String getName (){
  return this.name;
}

/*              WINDOW ACTIVATIONS                 */
/**
  * The window listener methods
  * @param event (WindowEvent)
**/
public void windowClosing        (WindowEvent event){System.exit(0);}
public void windowActivated      (WindowEvent event){}
public void windowClosed         (WindowEvent event){}
public void windowDeactivated    (WindowEvent event){}
public void windowDeiconified    (WindowEvent event){}
public void windowIconified      (WindowEvent event){}
public void windowOpened         (WindowEvent event){}

/**
  * button okay action performed
  * Takes the information entered for the operator. Creates a new operator and
  * calls add or change method of the draw class. And sends new operator
  * information there to be added to the list.
  * @param e (ActionEvent)
**/
void btnOK_actionPerformed(ActionEvent e) {
  if(chkResult.isSelected()){
     display = true;
  }
```

```
   else{
     display = false;
   }
   setName(txtName.getText());

   if (noOfInputNodes > 3)
      inputNodes[3] = getInput4();
   if (noOfInputNodes > 2)
      inputNodes[2] = getInput3();
   if (noOfInputNodes > 1)
      inputNodes[1] = getInput2();
   if (noOfInputNodes > 0)
      inputNodes[0] = getInput1();

   buildQuery();
   System.out.println(query);
   test.window.cnvDFQL.add
                  (x,y,getName(),getnumberOfNodes(),inputNodes,display,
                   getOperatorType(), query);
   this.dispose();
 }

/**
  * Loads the relations with the tables of the selected database and previously
  * created operators.
**/
void loadSelect() {
  Vector deneme = test.window.currentDatabase.getTableNames();
  for (int index = 0 ; index < deneme.size(); index++) {
     cmbInput1.addItem((String) deneme.elementAt(index));
     cmbInput2.addItem((String) deneme.elementAt(index));
  }

  for (int xi = 0; xi <test.window.cnvDFQL.numOfOperators;xi++){
     cmbInput1.addItem(test.window.cnvDFQL.operators[xi].name);
     cmbInput2.addItem(test.window.cnvDFQL.operators[xi].name);
  }
  cmbInput1.removeItem(name);
  cmbInput2.removeItem(name);
}

/**
  * button cancel action performed
  * terminates this frame without creating an operator. decrements the counter
  * @param e (ActionEvent)
**/
```

```
void btnCancel_actionPerformed(ActionEvent e) {
  this.dispose();
}

/**
 * This portion of the code produces the SQL query for this operator according
 * to the information entered by user.
**/
public void buildQuery(){

  StringBuffer skeleton        = new StringBuffer(querySkeleton);
  StringBuffer result;
  String temp;
  String tempToken             = "";
  int inputCount               = 0;
  int lastIndex                = 0;
  String checkRelation         = null;
  boolean last                 = false;

  // Check the relation slots in the query skeleton and replace them with the
      names
  // of the relations entered by the user
  for (int relIndex = 1; relIndex <= noOfRelations; relIndex++){
    temp = "Rel" + relIndex;
    result = new StringBuffer();
    for (int bufferIndex = 0; bufferIndex < skeleton.length()-3; bufferIndex++){
      tempToken = skeleton.substring(bufferIndex, bufferIndex+4);
      if (temp.equalsIgnoreCase(tempToken)){
        checkRelation = isOtherOperator (inputNodes[inputCount]);
        if (bufferIndex == skeleton.length()-4){
          last = true;
        }
        if (checkRelation == null){
          result = result.append(inputNodes[inputCount]);
        }
        else{
          result = result.append("("+ checkRelation+ ")");
        }
        bufferIndex = bufferIndex + 3;
      }
      else{
        result = result.append(skeleton.charAt(bufferIndex));
      }

    }
    if (!last){
      result = result.append(skeleton.substring(skeleton.length()-3,
                        skeleton.length()));
    }
    else{
      last = false;
    }
    inputCount++;
    skeleton = result;
  }

  // Check the condition slots in the query skeleton and replace them with
  // the names of the conditions entered by the user
  for (int conIndex = 1; conIndex <= noOfConditions; conIndex++){
    result = new StringBuffer();
    temp = "Con" + conIndex;
    for (int bufferIndex = 0; bufferIndex < skeleton.length()-3; bufferIndex++){
      tempToken = skeleton.substring(bufferIndex, bufferIndex+4);
      if (temp.equalsIgnoreCase(tempToken)){
        if (bufferIndex == skeleton.length()-4){
          last = true;
        }
        result = result.append(inputNodes[inputCount]);
        bufferIndex = bufferIndex + 3;
      }
      else{
        result = result.append(skeleton.charAt(bufferIndex));
      }
      lastIndex = bufferIndex+1;
    }
    if (!last){
      result = result.append(skeleton.substring(lastIndex,skeleton.length()));
    }
    else{
      last = false;
    }
    inputCount++;
    skeleton = result;
  }

  // Check the attribute slots in the query skeleton and replace them with the
      names
  // of the attribute names entered by the user
  for (int attIndex = 1; attIndex <= noOfAttributes; attIndex++){
    temp = "Att" + attIndex;
    result = new StringBuffer();
    for (int bufferIndex = 0; bufferIndex < skeleton.length()-3; bufferIndex++){
          tempToken = skeleton.substring(bufferIndex, bufferIndex+4);
          if (temp.equalsIgnoreCase(tempToken)){
            if (bufferIndex == skeleton.length()-4){
              last = true;
            }
            result = result.append(inputNodes[inputCount]);
            bufferIndex = bufferIndex + 3;
          }
          else{
            result = result.append(skeleton.charAt(bufferIndex));
          }
        }
        if (!last){
          result = result.append(skeleton.substring(skeleton.length()-3,
                        skeleton.length()));
        }
        else{
          last = false;
        }
        inputCount++;
        skeleton = result;
      }
      this.query = skeleton.toString();
    }

    /**
     * This portion of the code looks for the type of the relations, if they are
     * tables it returns null, otherwise if is another operator, it returns the
     * query of the operator which the current operator related to.
    **/
    public String isOtherOperator (String type) {
      String result = null;
      for (int index = 0; index < test.window.DFQLQuery.size(); index++) {
        Operator temp = (Operator) test.window.DFQLQuery.elementAt(index);
        if (type.equalsIgnoreCase(temp.name)) {
          result = temp.query;
          break;
        }
      }
     return result;
    }
  }
```

27. PrintPreview.Java

```
package ThesisGUI;

import java.awt.*;
import javax.swing.*;
import java.awt.print.*;

public class PrintPreview implements Printable {
 private Component componentToBePrinted;

 public static void printComponent(Component c) {
  new PrintPreview(c).print();
 }

 public PrintPreview(Component componentToBePrinted) {
  this.componentToBePrinted = componentToBePrinted;
 }

 public void print() {
  PrinterJob printJob = PrinterJob.getPrinterJob();
  printJob.setPrintable(this);
  if (printJob.printDialog())
   try {
    printJob.print();
   } catch(PrinterException pe) {
    System.out.println("Error printing: " + pe);
   }
}

public int print(Graphics g, PageFormat pageFormat, int pageIndex) {
 if (pageIndex > 0) {
  return(NO_SUCH_PAGE);
 }
 else {
  Graphics2D g2d = (Graphics2D)g;
  g2d.translate(pageFormat.getImageableX(), pageFormat.getImageableY());
  disableDoubleBuffering(componentToBePrinted);
  componentToBePrinted.paint(g2d);
  enableDoubleBuffering(componentToBePrinted);
  return(PAGE_EXISTS);
 }
}
```

```
 /** The speed and quality of printing suffers dramatically if
  *  any of the containers have double buffering turned on.
  *  So this turns if off globally.
  *  @see enableDoubleBuffering
  */
 public static void disableDoubleBuffering(Component c) {
  RepaintManager currentManager = RepaintManager.currentManager(c);
  currentManager.setDoubleBufferingEnabled(false);
 }

 /** Re-enables double buffering globally. */
 public static void enableDoubleBuffering(Component c) {
  RepaintManager currentManager = RepaintManager.currentManager(c);
  currentManager.setDoubleBufferingEnabled(true);
 }
}
```

28. Relation.Java

```
package ThesisGUI;

import java.util.*;
import javax.swing.table.*;

/**
 * This class implements the necessary model JTable, so the JTable object and
 * display the rows of data
 *
 * AUTHOR : BA & IS
**/
public class Relation extends AbstractTableModel{
 Vector columnNames;
 Vector rows;

 // Default constructor
 public Relation (){
  rows                 = new Vector();
  columnNames          = new Vector();
 }

 // Assigns the table content and the column heading
 public void assignData(Vector tableData, Vector columnHeader) {
  this.rows = tableData;
  this.columnNames = columnHeader;
  fireTableChanged(null);
 }
```

```
  // Table Model Implementation
  public String getColumnName(int column) {
    String result = "";

    if (columnNames.elementAt(column) != null) {
      Attribute temp      = (Attribute)columnNames.elementAt(column);
      result              = temp.getAttributeName();
    }
    return result;
  }

  public int getColumnCount() {
    return columnNames.size();
  }

  public int getRowCount() {
    return rows.size();
  }

  public Object getValueAt (int aRow, int aColumn) {
    Vector row = (Vector)rows.elementAt(aRow);
    return row.elementAt(aColumn);
  }

  public void clear (){
    this.rows.removeAllElements();
    this.columnNames.removeAllElements();
    fireTableChanged(null);
  }
}
```

29. SaveConfig.Java

```
package ThesisGUI;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;

/**
  * This frame is designed to be used to save a query,
  * that is created by the user, for future use.
  * AUTHOR : IS & BA
**/
```

```
public class saveConfig extends JFrame {
 /** File Dialog object for saving a file **/
 FileDialog file = new FileDialog(this,"Save File",FileDialog.SAVE);

 /**
  * Boolean TRUE  : if user wants to save a user defined operator
  *        FALSE : if user wants to save a DFQL Query
 **/
 boolean userDefined = false;

 // The fields required in order to save a user defined operator
 int xCoord, yCoord, relNo, conNo, attNo;
 String querySkel, opType;

 /*              CONSTRUCTORS                    */
 /**
  * The default constructor
 **/
 public saveConfig() {
  try {
   jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }
 }

 /**
  * The constructor to save a user defined operator
 **/
 public saveConfig(int x, int y, int rel, int cond, int att, String skeleton,
                        String type) {
  try {
   userDefined       = true;
   xCoord            = x;
   yCoord            = y;
   relNo             = rel;
   conNo             = cond;
   attNo             = att;
   querySkel         = skeleton;
   opType            = type;
   jbInit();
  }
  catch(Exception e) {
   e.printStackTrace();
  }}
```

```
/*              INITIATE METHOD                    */
/**
  * The initiation
  * creates the file dialog object. And inititates it.
  * Later on calls save config method.
**/
private void jbInit() throws Exception {
  this.setResizable (false);

  // If user wants to save a user defined operator, set the file type to ".uso"
  if (userDefined){
    file.setDirectory ("d:\\JBuilder3\\myprojects\\ThesisGUI\\operator\\");
    file.setFile ("*.uso");
    saveDesign   ();
  }

  // If user wants to save a DFQL query, set the file type to ".dfql"
  else {
    file.setDirectory ("d:\\JBuilder3\\myprojects\\ThesisGUI\\");
    file.setFile ("*.dfql");
    saveConfig   ();
  }
}

/**
  * Checks the name of the file. If a file is entered. It starts to write
  * the operator data to file. It repeats the same procedure
  * for every operator created in the DFQL query to be saved.
**/
void saveConfig(){
  file.show(true);
  String fileName = file.getFile();
  if (fileName!=null){
    try {
      File newFile = new File (file.getDirectory()+ fileName);
      FileWriter outStream = new FileWriter (newFile);
      BufferedWriter bufFile = new BufferedWriter (outStream);

      bufFile.write(test.window.DFQLQuery.size()+"\n");

      for(int xi =0;xi<test.window.DFQLQuery.size();xi++){
        Operator temp = (Operator)test.window.DFQLQuery.elementAt(xi);
        bufFile.write(temp.type+"\n");
        bufFile.write(temp.name+"\n");
        bufFile.write(temp.x+"\n");
        bufFile.write(temp.y+"\n");
        bufFile.write(temp.numberOfInputNodes+"\n");
        bufFile.write(temp.display+"\n");
        bufFile.write(temp.query+"\n");

        for(int fi=0;fi<temp.numberOfInputNodes;fi++) {
          bufFile.write(temp.input[fi]+"\n");
        }
      }
      bufFile.close();
    }
    catch (IOException ex){}
  }
}

/**
 * Checks the name of the file. If a file is entered. It starts to write
 * the user defined operator data to file.
**/
void saveDesign(){
 file.show(true);
 String fileName = file.getFile();
 if (fileName!=null){
  try {
    File newFile = new File (file.getDirectory()+fileName);
    FileWriter outStream = new FileWriter (newFile);
    BufferedWriter bufFile = new BufferedWriter (outStream);

    bufFile.write(xCoord+"\n");
    bufFile.write(yCoord+"\n");
    bufFile.write(relNo+"\n");
    bufFile.write(conNo+"\n");
    bufFile.write(attNo+"\n");
    bufFile.write(querySkel+"\n");
    bufFile.write(opType+"\n");
    bufFile.close();
  }
  catch (IOException ex){}
 }
}
}
```

30. Settings.Java

```
package ThesisGUI;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;

/**
 * This class is used to provide the user an interface to change performance
 * related attributes of the connected database. There are three important settings
 * for the database connection: Store, AskDB, and UpdatePeriod
 * @author BA & IS
 * @see Database-store
 * @see Database-askDB
 * @see Database-updatePeriod
**/
public class Settings extends JFrame {

  JCheckBox checkBoxStore          = new JCheckBox();
  JCheckBox checkBoxAskDB          = new JCheckBox();
  JLabel labelUpdateTime           = new JLabel();
  JTextField txtUpdateTime         = new JTextField();
  JButton buttonOK                 = new JButton("OK");
  JButton buttonCancel             = new JButton("Cancel");

  private Database setDatabase;

  public Settings(Database currentDatabase) {
    try {
      setDatabase = currentDatabase;
      jbInit();
    }
    catch(Exception e) {
      e.printStackTrace();
    }
  }

  private void jbInit() throws Exception {
    this.setResizable(false);
    this.getContentPane().setLayout(null);
    this.setBounds(new Rectangle(250,180,270,200));
    this.setTitle("Settings - "+ setDatabase.getURL());

    checkBoxStore.setHorizontalTextPosition(SwingConstants.LEADING);
    checkBoxStore.setText("Store Executed Queries     ");
 checkBoxStore.setHorizontalAlignment(SwingConstants.LEADING);
 checkBoxStore.setBounds  (new Rectangle(32, 9, 173, 25));
 checkBoxStore.setSelected (setDatabase.isStore());
 checkBoxStore.setBorder   (BorderFactory.createEtchedBorder());

 checkBoxAskDB.setHorizontalTextPosition(SwingConstants.LEADING);
 checkBoxAskDB.setText   ("Get results from Database");
 checkBoxAskDB.setHorizontalAlignment(SwingConstants.LEADING);
 checkBoxAskDB.setBounds         (new Rectangle(32, 47, 173, 25));
 checkBoxAskDB.setSelected       (setDatabase.isAskDB());
 checkBoxAskDB.setBorder         (BorderFactory.createEtchedBorder());

 labelUpdateTime.setText         ("Enter Update Time (mins)");
 labelUpdateTime.setBounds       (new Rectangle(35, 92, 143, 25));

 txtUpdateTime.setBounds         (new Rectangle(180, 94, 47, 20));
 txtUpdateTime.setText(Integer.toString(setDatabase.getUpdateTime()));

 buttonOK.setText("OK");
 buttonOK.setBounds(new Rectangle(70,135,90,20));
 buttonOK.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   buttonOK_actionPerformed(e);
  }});

 buttonCancel.setText("Cancel");
 buttonCancel.setBounds ((new Rectangle(170,135,90,20)));
 buttonCancel.addActionListener(new java.awt.event.ActionListener() {
  public void actionPerformed(ActionEvent e) {
   buttonCancel_actionPerformed(e);
  }});

 this.getContentPane().add(checkBoxStore, null);
 this.getContentPane().add(labelUpdateTime, null);
 this.getContentPane().add(checkBoxAskDB, null);
 this.getContentPane().add(txtUpdateTime, null);
 this.getContentPane().add(buttonOK, null);
 this.getContentPane().add(buttonCancel, null);
}

void buttonOK_actionPerformed(ActionEvent e) {
 setDatabase.setStore(checkBoxStore.isSelected());
 setDatabase.setAskDB(checkBoxAskDB.isSelected());
 setDatabase.setUpdatePeriod(Integer.parseInt(txtUpdateTime.getText()));
 this.dispose();
}
```

```
  void buttonCancel_actionPerformed(ActionEvent e) {
    this.dispose();
  }
}
```

31. Table.Java

```
package ThesisGUI;

import java.util.*;
import java.util.Date;
import java.lang.*;
import java.sql.*;

/**
  * This class holds the information (Relation name, data, attributes, owner,
  * query to get this relation) of a relation.
  * @author BA
**/
public class Table{

  /*  DATA MEMBERS   */

  /**
    * Updates the query data after this period
  **/
  private int consTime;

  /**
    * Relation Name
  **/
  private String name;

  /**
    * Attributes of this relation
  **/
  private Vector attributes;

  /**
    * Owner Database of this relation
  **/
  private Database owner;
```

```
/**
 * The relation's data
**/
private Vector data;

/**
 * Query to get this relation
**/
private String query;

/**
 * The time when the query is executed
**/
private Date tStamp;

/* CONSTRUCTORS */
/**
 * Constructor
 * @param query1 the query to get this relation
 * @dBase Owner of this relation (Database object)
 * @see Database
**/
protected Table (String query1, Database dBase) throws SQLException{
   name        = null;            // Name of the relation
   owner       = dBase;           // Owner database of this relation
   attributes  = new Vector();    // Attributes of this relation
   data        = new Vector();    // Data of this relation
   query       = query1;          // Query to get this relation
   tStamp      = new Date();      // Timestamp

  // finds and stores the attributes of this relation
   initiateRelationAttributes();

   // stores the relation data in the data vector
   initiateData();
}

/* PUBLIC METHODS */
/**
 * @return the table name (String)
**/
public String getTableName(){
  return name;
}
```

```
/**
  * @return The relation's attributes (Vector of Attribute Class)
  * @see Attribute Class
**/
public Vector getAttributes(){
  return attributes;
}

/**
  * @return The relation's attribute names (Vector of String)
**/
public Vector getAttributeNames (){
  Vector attNames = new Vector();
  for (int attIndex = 0; attIndex < attributes.size(); attIndex++) {
    Attribute att = (Attribute) attributes.elementAt(attIndex);
    attNames.add(att.getAttributeName());
  }
  return attNames;
}

/**
  * @return The relation data (Vector)
**/
public Vector getData() throws SQLException{
  return data;
}

/**
  * @return the query to get this relation (String)

**/
public String getQuery(){
 return query;
}

public Attribute getAttData (String attName) {
  Attribute result = null;
  for (int attIndex = 0; attIndex < attributes.size(); attIndex++) {
    result = (Attribute) attributes.elementAt(attIndex);
    if (result.getAttributeName().equalsIgnoreCase(attName)) {
      break;
    }
  }
  return result;
}
```

```
/* PRIVATE METHODS   */
/**
  * Stores the Attribute information of the current query
**/
private void initiateRelationAttributes()throws SQLException{

  ResultSetMetaData metaData       = owner.getQueryMetaData();
  int resultColumnCount            = metaData.getColumnCount();

  for (int columnIndex = 1; columnIndex <= resultColumnCount;
      columnIndex++){
    Attribute temp = new Attribute();
    temp.setAttributeName (metaData.getColumnName(columnIndex));
    temp.setAttributeType  (metaData.getColumnTypeName(columnIndex));
    temp.setAttributeSize  (metaData.getColumnDisplaySize(columnIndex));
    temp.setNullable       (metaData.isNullable(columnIndex));
    temp.setSearchable     (metaData.isSearchable(columnIndex));
    temp.setCaseSensitive  (metaData.isCaseSensitive(columnIndex));
    attributes.add         (temp);
  }
}

/**
  * Puts the relation's data into a vector of vectors.
**/
private void initiateData() throws SQLException{
  data = new Vector (); // vector of records
  Vector oneRow;        // a vector of one record
  int numberOfColumns = attributes.size();

  ResultSet temp = owner.getQueryResultSet();
  while (temp.next()) {

    oneRow = new Vector();

    // insert the record data into a record vector
    for (int i = 1; i <= numberOfColumns; i++) {
      oneRow.addElement(temp.getString(i));
    }

    // insert the record vector into data vector
    data.addElement(oneRow);
  }
}
```

```
    /* PROTECTED METHODS  */
    /**
      * @return the date of the query execution (Date object)
    **/
    protected Date getTimeStamp () {
      return tStamp;
    }

    /**
      * Sets the relation name
      * @param tableName the name of the relation (String)
    **/
    protected void setTableName (String tableName) {
      name = tableName;
    }

    /**
      * Synchronizes the data with the database by calling the 'initiateData()'
      * method and resets the time stamp
    **/
    protected void updateRelationData () throws SQLException{
      data = null;
      initiateData();  // Re-initiates the relation data
      tStamp = new Date();
    }
}
```

32. Test.Java

```
package ThesisGUI;

/**
  * creates a MainTest class and shows it.
  * AUTHOR : BA & IS
**/
public class test
{
  public static MainTest window = new MainTest();
  public static void main(String args[])
  {
    window.show();
  }
}
```

33. UserOperator.Java

```
package ThesisGUI;

import java.awt.*;
import javax.swing.*;
import java.util.*;
import com.borland.jbcl.control.*;
import java.awt.event.*;

/**
 * This class is used to display previously created user defined operators.
 * It allows user to use these operators as normal operators.
 * AUTHOR : IS & BA
**/
public class UserOperator extends JFrame {
 JTextField txtNoOfRelation        = new JTextField();
 JTextField txtNoOfCondition       = new JTextField();
 JTextField txtNoOfAttribute       = new JTextField();
 JTextField txtUserName            = new JTextField();
 JComboBox comboOperators          = new JComboBox();
 JButton buttonAdd                 = new JButton("Add");
 JButton buttonOK                  = new JButton("Ok");
 JButton buttonCancel              = new JButton("Cancel");
 JButton buttonProceed             = new JButton("Proceed ==>");

 JLabel lblOpName                  = new JLabel();
 JLabel lblRelation                = new JLabel();
 JLabel lblCondition               = new JLabel();
 JLabel lblAttribute               = new JLabel();
 JTextArea txtAreaOperators        = new JTextArea();

 int noOfRelation;
 int noOfCondition;
 int noOfAttribute;
 int noOfOperators;
 int x;
 int y;
 Vector operators;
 Vector queries;
 String query;

 /*          CONSTRUCTOR                */
 /**
  * Default constructor
 **/
```

```
public UserOperator() {
  try  {
    this.x = 100;
    this.y = 100;
    jbInit();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}

/*              CONSTRUCTOR                    */
/**
  * Constructor
  * @parameters X and Y (integer) coordinates of the operator
**/
public UserOperator(int x, int y) {
  try  {
    this.x = x;
    this.y = y;

    jbInit();
  }
  catch(Exception e) {
    e.printStackTrace();
  }
}

/*              INITIATE METHOD                    */
/**
  * The initiation
  * creates the frame for user to enter the operator data, and adds
  * buttons, labels, text area and combo boxes.
**/
private void jbInit() throws Exception {
  this.setSize  (490,240);
  this.setTitle ("User Defined Operator");

  lblOpName.setText          ("Name");
  lblRelation.setText        ("# Relation");
  lblCondition.setText       ("# Condition");
  lblAttribute.setText       ("# Attribute");

  lblOpName.setBounds        (new Rectangle(12, 15, 95, 28));
  lblRelation.setBounds      (new Rectangle(12, 50, 95, 28));
  lblCondition.setBounds     (new Rectangle(12, 86, 95, 28));
```

```
lblAttribute.setBounds      (new Rectangle(12, 121, 95, 28));
buttonOK.setBounds          (new Rectangle(308, 169, 72, 19));
buttonAdd.setBounds         (new Rectangle(369, 50, 90, 19));
buttonCancel.setBounds      (new Rectangle(383, 169, 78, 19));

txtNoOfRelation.setNextFocusableComponent(txtNoOfCondition);
txtNoOfRelation.setBounds           (new Rectangle(105, 48, 36, 29));
txtNoOfCondition.setNextFocusableComponent(txtNoOfAttribute);
txtNoOfCondition.setBounds          (new Rectangle(105, 84, 36, 29));
txtNoOfAttribute.setNextFocusableComponent(buttonProceed);
txtNoOfAttribute.setBounds          (new Rectangle(105, 120, 36, 29));
txtUserName.setNextFocusableComponent(txtNoOfRelation);
txtUserName.setBounds               (new Rectangle(105, 12, 84, 28));
comboOperators.setNextFocusableComponent(buttonAdd);
comboOperators.setBounds            (new Rectangle(310, 18, 156, 28));
buttonProceed.setNextFocusableComponent(comboOperators);
buttonProceed.setBounds             (new Rectangle(170, 75, 110, 19));
txtAreaOperators.setBounds          (new Rectangle(313, 73, 148, 91));

comboOperators.addItem   ("Select");
comboOperators.addItem   ("Join");
comboOperators.addItem   ("Project");
comboOperators.addItem   ("Intersect");
comboOperators.addItem   ("Diff");
comboOperators.addItem   ("Union");
comboOperators.addItem   ("GroupCnt");
comboOperators.addItem   ("Eqjoin");
comboOperators.addItem   ("GrpMin");
comboOperators.addItem   ("GrpMax");
comboOperators.addItem   ("GrpAvg");
comboOperators.addItem   ("GrpAllSat");

buttonOK.setEnabled       (false);
buttonAdd.setEnabled      (false);
buttonAdd.setNextFocusableComponent(comboOperators);
buttonAdd.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  buttonAdd_actionPerformed(e);
 }});

buttonOK.addActionListener(new java.awt.event.ActionListener() {
 public void actionPerformed(ActionEvent e) {
  buttonOK_actionPerformed(e);
 }});
```

```
  buttonCancel.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(ActionEvent e) {
    buttonCancel_actionPerformed(e);
   }});

  buttonProceed.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(ActionEvent e) {
    buttonProceed_actionPerformed(e);
   }});

  this.getContentPane().setLayout(null);
  txtUserName.requestFocus();

  this.getContentPane().add(txtUserName, null);
  this.getContentPane().add(txtNoOfCondition, null);
  this.getContentPane().add(txtNoOfAttribute, null);
  this.getContentPane().add(txtNoOfRelation, null);
  this.getContentPane().add(lblCondition, null);
  this.getContentPane().add(lblRelation, null);
  this.getContentPane().add(lblOpName, null);
  this.getContentPane().add(comboOperators, null);
  this.getContentPane().add(lblAttribute, null);
  this.getContentPane().add(buttonAdd, null);
  this.getContentPane().add(txtAreaOperators, null);
  this.getContentPane().add(buttonProceed, null);
  this.getContentPane().add(buttonCancel, null);
  this.getContentPane().add(buttonOK, null);

  noOfOperators      = 0;
  operators          = new Vector();
  queries            = new Vector();
  show(true);
 }

 /**
  * This function is used to select operators which will be used as a part
  * of user defined operator. Once the operator selected user presses add
  * button to create it.
 **/
 void buttonAdd_actionPerformed(ActionEvent e) {
  txtAreaOperators.append(comboOperators.getSelectedItem()+ "\n");
  operators.addElement(comboOperators.getSelectedItem());
  test.window.designOps.addElement("Op"+ operators.size());
  processOperators((String)comboOperators.getSelectedItem());
 }
```

```
/**
 * This function cancels all new operator creating process.
**/
void buttonCancel_actionPerformed(ActionEvent e) {
 this.dispose();
 test.window.setDesign(false);
}

/**
 * After the operator completely created. User presses this button to save it
 * and to leave this screen.
**/
void buttonOK_actionPerformed(ActionEvent e) {
  processQueries();
  saveConfig deneme = new saveConfig(x,y, noOfRelation,noOfCondition,
                             noOfAttribute, query,
                             txtUserName.getText());
  this.dispose();
  test.window.setDesign(false);
}

/**
 * When the operator is selected to be used as a part of user defined operator
 * the operator is created here.
**/
private void processOperators(String newOp){
   if (newOp.equalsIgnoreCase("select")){
     OperatorSelect newSelect = new OperatorSelect((String)test.window.
                                     designOps.lastElement());
   }
   else if (newOp.equalsIgnoreCase("project")){
     OperatorProject newProject = new OperatorProject((String)test.window.
                                     designOps.lastElement());
   }
   else if (newOp.equalsIgnoreCase("join")){
     OperatorJoin newJoin = new OperatorJoin((String)test.window.
                                designOps.lastElement());
   }
   else if (newOp.equalsIgnoreCase("union")){
     OperatorUnion newUnion = new OperatorUnion((String)test.window.
                                  designOps.lastElement());
   }
   else if (newOp.equalsIgnoreCase("intersect")){
     OperatorIntersect newIntersect = new OperatorIntersect((String)test.
                                          window.designOps.
                                          lastElement());
```

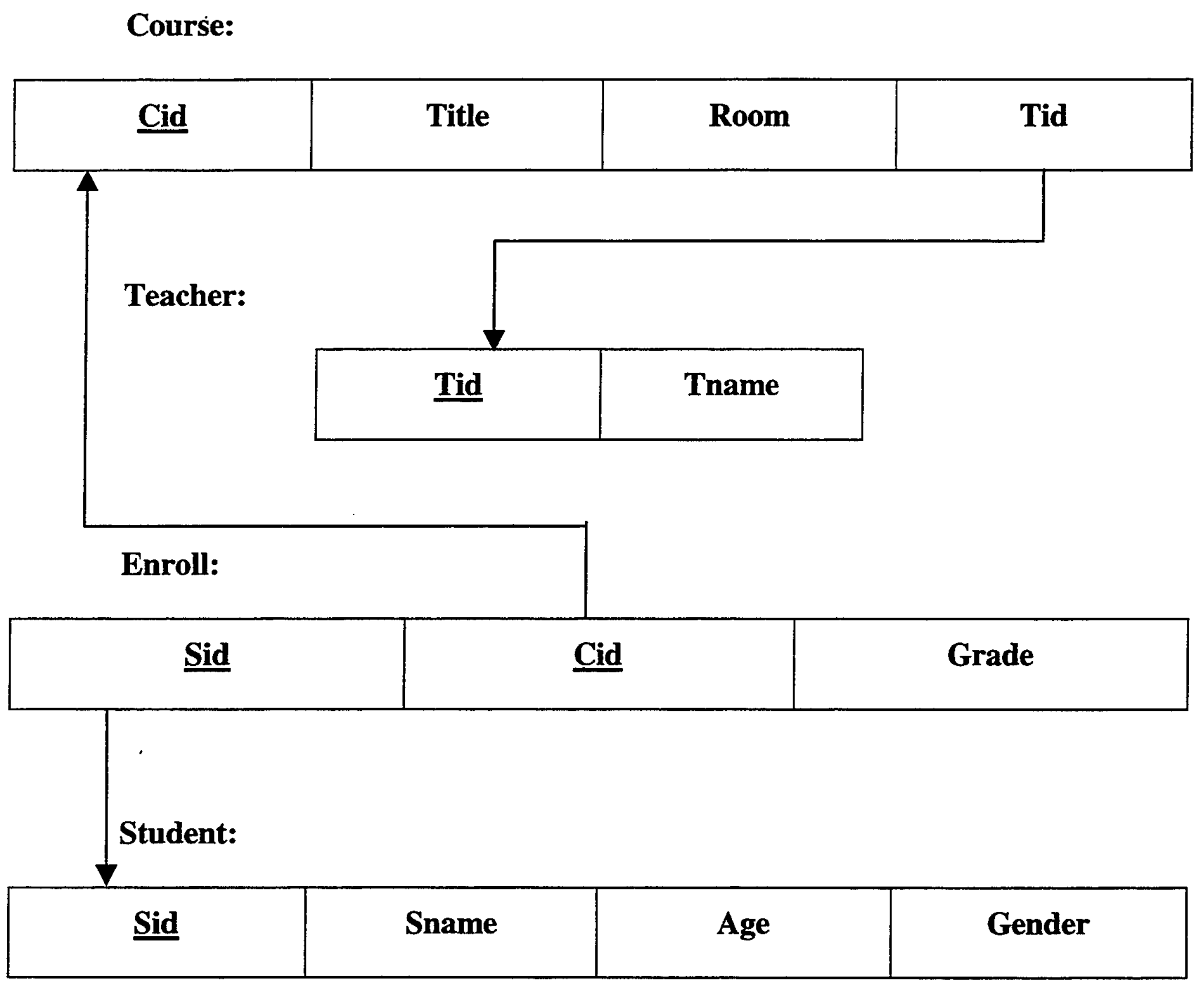
```
    }
    else if (newOp.equalsIgnoreCase("diff")){
      OperatorDiff newDiff = new OperatorDiff((String)test.window.
                                     designOps.lastElement());
    }
    else if (newOp.equalsIgnoreCase("eqjoin")){
      OperatorEqjoin newEqjoin = new OperatorEqjoin((String)test.window.
                                     designOps.lastElement());
    }
    else if (newOp.equalsIgnoreCase("groupcnt")){
      OperatorGroupCnt newGrpCnt = new OperatorGroupCnt
                             ((String)test.window.designOps.lastElement());
    }
    else if (newOp.equalsIgnoreCase("grpallsat")){
      OperatorGrpAllSat newGrpAllSat = new OperatorGrpAllSat
                             ((String)test.window. designOps.lastElement());
    }
    else if (newOp.equalsIgnoreCase("grpavg")){
      OperatorGrpAvg newGrpAvg = new OperatorGrpAvg((String)test.window.
                                     designOps.lastElement());
    }
    else if (newOp.equalsIgnoreCase("grpmax")){
      OperatorGrpMax newGrpMax = new OperatorGrpMax((String)test.window.
                                     designOps.lastElement());
    }
    else if (newOp.equalsIgnoreCase("grpmin")){
      OperatorGrpMin newGrpMin = new OperatorGrpMin((String)test.window.
                                     designOps.lastElement());
    }
  }

void buttonProceed_actionPerformed(ActionEvent e) {
  test.window.designRelation.removeAllElements();
  test.window.designCondition.removeAllElements();
  test.window.designAttribute.removeAllElements();
  test.window.designOps.removeAllElements();

  noOfRelation        = Integer.parseInt(txtNoOfRelation.getText());
  noOfCondition       = Integer.parseInt(txtNoOfCondition.getText());
  noOfAttribute       = Integer.parseInt(txtNoOfAttribute.getText());

  for (int relIndex = 1; relIndex <= noOfRelation; relIndex++){
    test.window.designRelation.addElement("Rel"+relIndex);
  }

  for (int conIndex = 1; conIndex <= noOfCondition; conIndex++){
   test.window.designCondition.addElement("Con"+conIndex);
  }

  for (int attIndex = 1; attIndex <= noOfAttribute; attIndex++){
   test.window.designAttribute.addElement("Att"+attIndex);
  }

  buttonAdd.setEnabled(true);
  buttonOK.setEnabled(true);
 }

 /**
  * This function combines all the operators queries to produce the new operators
  * query.
 **/
 private void processQueries(){
  String currentQuery = (String)test.window.designQueries.lastElement();
  query = checkQuery(currentQuery);
  System.out.println(query);
 }

 /**
  * Query of the each operator is analyzed here and the connections
  * between the operators established.
 **/
 private String checkQuery(String queryCheck){
  String lastQuery = "";
  StringTokenizer queryTokenize = new StringTokenizer(queryCheck," ",true);
  String tryQuery;
  String queryPart;
  int queryIndex;
  while (queryTokenize.hasMoreTokens()){
   tryQuery = (String)queryTokenize.nextToken();
   queryIndex = isOperator(tryQuery);
   if (queryIndex > -1){
    queryPart = checkQuery((String)test.window.designQueries.elementAt
                (queryIndex));
    lastQuery = lastQuery + "( " + queryPart + " )";
   }
   else{
    lastQuery = lastQuery + tryQuery;
   }
  }
  return lastQuery;
 }
 /**
  * This function is used to identify the item if it is an operator or not.
 **/
 private int isOperator(String op){
   for (int opIndex=0; opIndex < test.window.designOps.size();opIndex++){
     if (op.equalsIgnoreCase
         ((String)test.window.designOps.elementAt(opIndex))){
       return opIndex;
     }
   }
   return -1;
 }

}
```

APPENDIX – B

EXAMPLE DATABASE

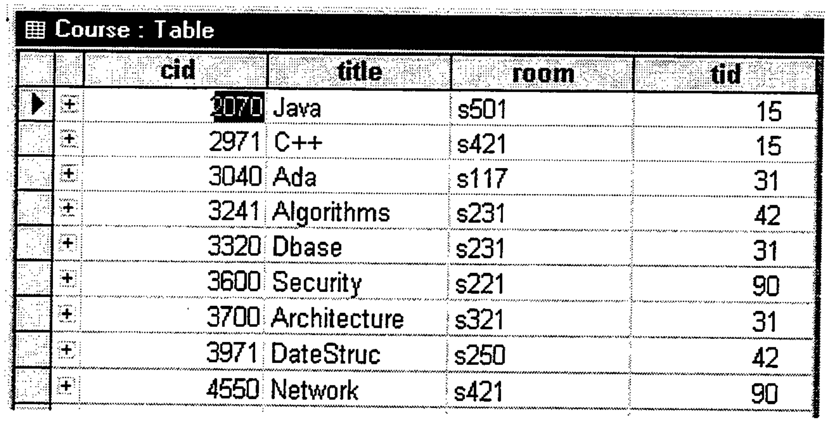
Through out this thesis all the query examples are matched with the relational schema database, which is called the STUDENT database.

RELATIONS



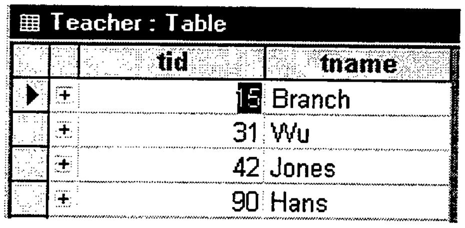
CONTENTS OF RELATIONS

Course:


Course : Table

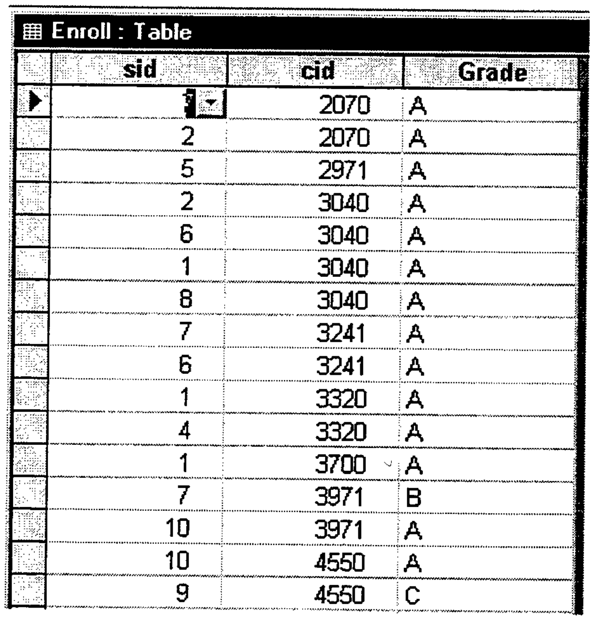
cid	title	room	tid
2070	Java	s501	15
2971	C++	s421	15
3040	Ada	s117	31
3241	Algorithms	s231	42
3320	Dbase	s231	31
3600	Security	s221	90
3700	Architecture	s321	31
3971	DateStruc	s250	42
4550	Network	s421	90

Teacher:


Teacher : Table

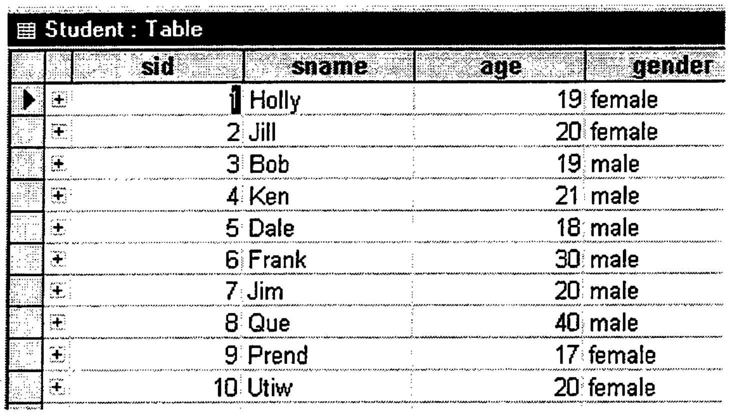
tid	tname
15	Branch
31	Wu
42	Jones
90	Hans

Enroll:


Enroll : Table

sid	cid	Grade
1	2070	A
2	2070	A
5	2971	A
2	3040	A
6	3040	A
1	3040	A
8	3040	A
7	3241	A
6	3241	A
1	3320	A
4	3320	A
1	3700	A
7	3971	B
10	3971	A
10	4550	A
9	4550	C

Student:



Student : Table

sid	sname	age	gender
1	Holly	19	female
2	Jill	20	female
3	Bob	19	male
4	Ken	21	male
5	Dale	18	male
6	Frank	30	male
7	Jim	20	male
8	Que	40	male
9	Prend	17	female
10	Utiw	20	female

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

1. Zloof, M.M., Query-by-Example: A Database Language, IBM System Journal, vol. 16, 1977.

2. Clark, Gard J., Wu, Thomas C., DFQL: Dataflow Query Language for Relational Databases, Information & Management 27 (1994) 1-15.

3. Ozsoyoglu, G., Matos, V. and Ozsoyoglu, Z. M., Query Processing Techniques in The Summary-Table-by-Example Database Query Language, ACM Transactions on Database Systems, vol. 14, no 4. December 1989.

4. Ozsoyoglu, G.,and Wang, H., Example-Based Graphical Database Query Languages, Computer, vol 26, no. 5, May 1993.

5. Ramakrishnan, R., and Gehrke, J., Database Management Systems, 2nd edition McGraw-Hill Higher Education Publishing Company, Inc., 1998.

6. Rob, P., and Coronel, C., Database Systems : Design, Implementation, and Management, Course Technology Publishing Company, 1999.

7. Anderson, J.A brief history of SQL, July 1995. (http://cesspool.crseo.ucsb.edu:8679/TUT1/hist_01.htm)

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 2
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Dr.D.C.Boger 4
   Chairman
   Department of Computer Science
   Code CS
   Naval Postgraduate School
   Monterey, CA 93943-5100

4. Prof.C.Thomas Wu 1
   Department of Computer Science
   Code CS/WG
   Naval Postgraduate School
   Monterey, CA 93943-5100

5. LCDR Christopher S. Eagle 1
   Department of Computer Science
   Code CS/CE
   Naval Postgraduate School
   Monterey, CA 93943-5100

6. Deniz Kuvvetleri Komutanligi 1
   Personel Tedarik ve Yetistirme Daire Baskanligi
   06100 Bakanliklar, Ankara
   TURKEY

7. Deniz Harp Okulu Komutanligi 1
   Kutuphane
   Tuzla, Istanbul 81704
   TURKEY

8. Baybora Aksoy 1
Osmaniye Fabrikalar Caddesi
Ilker Sitesi, C Blok, Daire 21
Bakirkoy, Istanbul
TURKEY

9. Ilker Sahin 1
Osmaniye Fabrikalar Caddesi
Ilker Sitesi, C Blok, Daire 21
Bakirkoy, Istanbul
TURKEY